CHAPTER

6

Convolution

Convolution is a mathematical way of combining two signals to form a third signal. It is the single most important technique in Digital Signal Processing. Using the strategy of impulse decomposition, systems are described by a signal called the *impulse response*. Convolution is important because it relates the three signals of interest: the input signal, the output signal, and the impulse response. This chapter presents convolution from two different viewpoints, called the input side algorithm and the output side algorithm. Convolution provides the mathematical framework for DSP; there is nothing more important in this book.

The Delta Function and Impulse Response

The previous chapter describes how a signal can be decomposed into a group of components called **impulses**. An impulse is a signal composed of all zeros, except a single nonzero point. In effect, impulse decomposition provides a way to analyze signals one sample at a time. The previous chapter also presented the fundamental concept of DSP: the input signal is decomposed into simple additive components, each of these components is passed through a linear system, and the resulting output components are synthesized (added). The signal resulting from this divide-and-conquer procedure is identical to that obtained by directly passing the original signal through the system. While many different decompositions are possible, two form the backbone of signal processing: impulse decomposition and Fourier decomposition. When impulse decomposition is used, the procedure can be described by a mathematical operation called **convolution**. In this chapter (and most of the following ones) we will only be dealing with *discrete* signals. Convolution also applies to *continuous* signals, but the mathematics is more complicated. We will look at how continious signals are processed in Chapter 13.

Figure 6-1 defines two important terms used in DSP. The first is the **delta function**, symbolized by the Greek letter delta, **δ[*n*]**. The delta function is a *normalized* impulse, that is, sample number zero has a value of one, while

all other samples have a value of zero. For this reason, the delta function is frequently called the **unit impulse**.

The second term defined in Fig. 6-1 is the **impulse response**. As the name suggests, the impulse response is the signal that exits a system when a delta function (unit impulse) is the input. If two systems are different in any way, they will have different impulse responses. Just as the input and output signals are often called $x[n]$ and $y[n]$, the impulse response is usually given the symbol, $\boldsymbol{h[n]}$. Of course, this can be changed if a more descriptive name is available, for instance, $f[n]$ might be used to identify the impulse response of a *filter*.

Any impulse can be represented as a *shifted* and *scaled* delta function. Consider a signal, $a[n]$, composed of all zeros except sample number 8, which has a value of -3. This is the same as a delta function shifted to the right by 8 samples, and multiplied by -3. In equation form: $a[n] = -3\delta[n-8]$. Make sure you understand this notation, it is used in nearly all DSP equations.

If the input to a system is an impulse, such as $-3\delta[n-8]$, what is the system's output? This is where the properties of homogeneity and shift invariance are used. Scaling and shifting the input results in an identical scaling and shifting of the output. If $\delta[n]$ results in $h[n]$, it follows that $-3\delta[n-8]$ results in $-3h[n-8]$. In words, the output is a version of the impulse response that has been *shifted* and *scaled* by the same amount as the delta function on the input. If you know a system's impulse response, you immediately know how it will react to *any* impulse.

Convolution

Let's summarize this way of understanding how a system changes an input signal into an output signal. First, the input signal can be decomposed into a set of impulses, each of which can be viewed as a scaled and shifted delta function. Second, the output resulting from each impulse is a scaled and shifted version of the impulse response. Third, the overall output signal can be found by adding these scaled and shifted impulse responses. In other words, if we know a system's impulse response, then we can calculate what the output will be for any possible input signal. This means we know *everything* about the system. There is nothing more that can be learned about a linear system's characteristics. (However, in later chapters we will show that this information can be represented in different forms).

The impulse response goes by a different name in some applications. If the system being considered is a *filter*, the impulse response is called the **filter kernel**, the **convolution kernel**, or simply, the **kernel**. In image processing, the impulse response is called the **point spread function**. While these terms are used in slightly different ways, they all mean the same thing, the signal produced by a system when the input is a delta function.

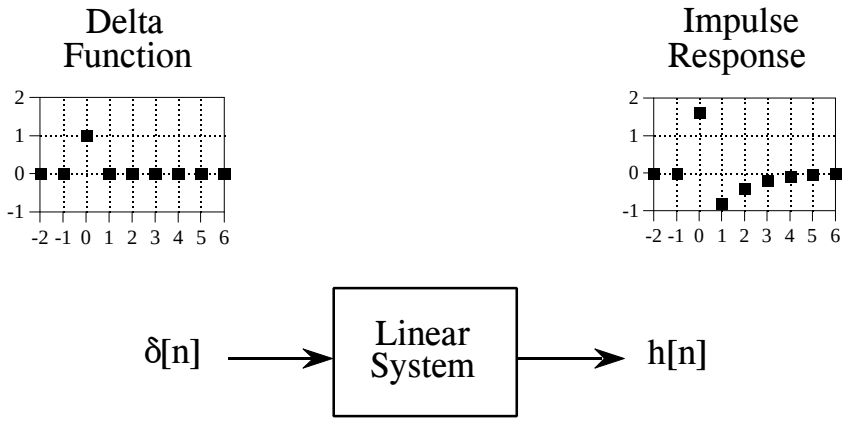
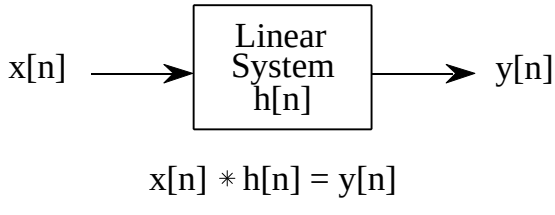


FIGURE 6-1
Definition of *delta function* and *impulse response*. The delta function is a normalized impulse. All of its samples have a value of zero, except for sample number zero, which has a value of one. The Greek letter delta, $\delta[n]$, is used to identify the delta function. The *impulse response* of a linear system, usually denoted by $h[n]$, is the output of the system when the input is a delta function.

Convolution is a formal mathematical operation, just as multiplication, addition, and integration. Addition takes two *numbers* and produces a third *number*, while convolution takes two *signals* and produces a third *signal*. Convolution is used in the mathematics of many fields, such as probability and statistics. In linear systems, convolution is used to describe the relationship between three signals of interest: the input signal, the impulse response, and the output signal.

Figure 6-2 shows the notation when convolution is used with linear systems. An input signal, $x[n]$, enters a linear system with an impulse response, $h[n]$, resulting in an output signal, $y[n]$. In equation form: $x[n] \ast h[n] = y[n]$. Expressed in words, the input signal convolved with the impulse response is equal to the output signal. Just as addition is represented by the plus, +, and multiplication by the cross, ×, convolution is represented by the star, ✳. It is unfortunate that most programming languages also use the star to indicate multiplication. A star in a computer program means multiplication, while a star in an equation means convolution.

FIGURE 6-2
How convolution is used in DSP. The output signal from a linear system is equal to the input signal *convolved* with the system's impulse response. Convolution is denoted by a star when writing equations.



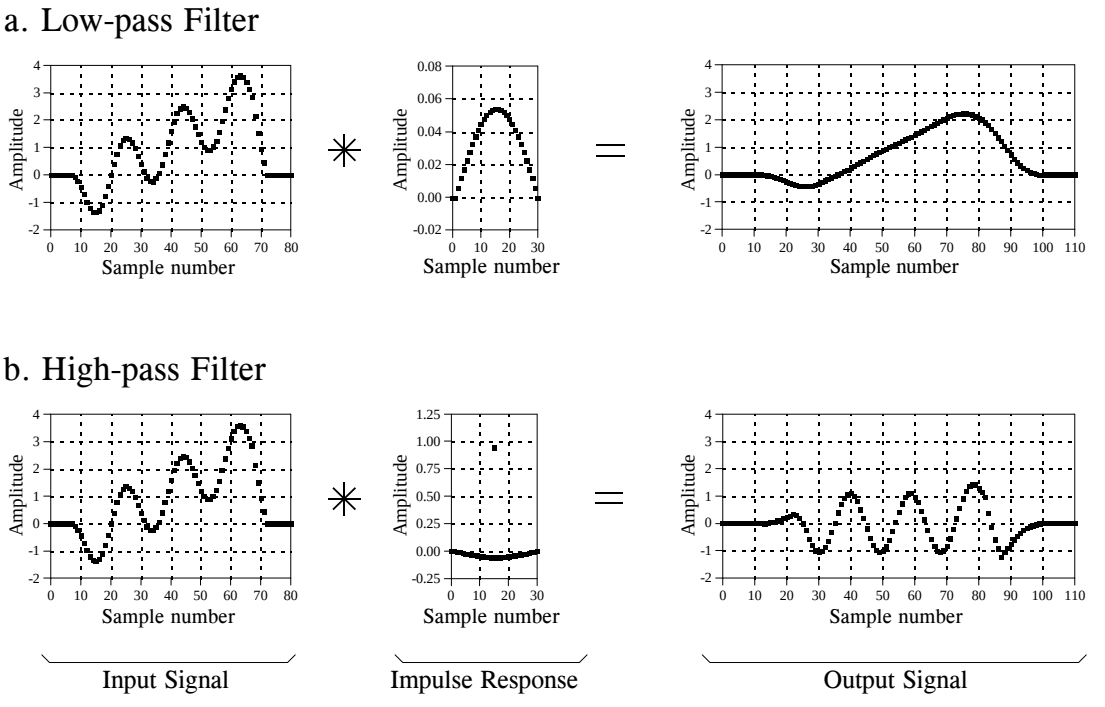


FIGURE 6-3
Examples of low-pass and high-pass filtering using convolution. In this example, the input signal is a few cycles of a sine wave plus a slowly rising ramp. These two components are separated by using properly selected impulse responses.

Figure 6-3 shows convolution being used for low-pass and high-pass filtering. The example input signal is the sum of two components: three cycles of a sine wave (representing a high frequency), plus a slowly rising ramp (composed of low frequencies). In (a), the impulse response for the low-pass filter is a smooth arch, resulting in only the slowly changing ramp waveform being passed to the output. Similarly, the high-pass filter, (b), allows only the more rapidly changing sinusoid to pass.

Figure 6-4 illustrates two additional examples of how convolution is used to process signals. The inverting attenuator, (a), flips the signal top-for-bottom, and reduces its amplitude. The discrete derivative (also called the first difference), shown in (b), results in an output signal related to the *slope* of the input signal.

Notice the lengths of the signals in Figs. 6-3 and 6-4. The input signals are 81 samples long, while each impulse response is composed of 31 samples. In most DSP applications, the input signal is hundreds, thousands, or even millions of samples in length. The impulse response is usually much shorter, say, a few points to a few hundred points. The mathematics behind convolution doesn't restrict how long these signals are. It does, however, specify the length of the output signal. The length of the output signal is

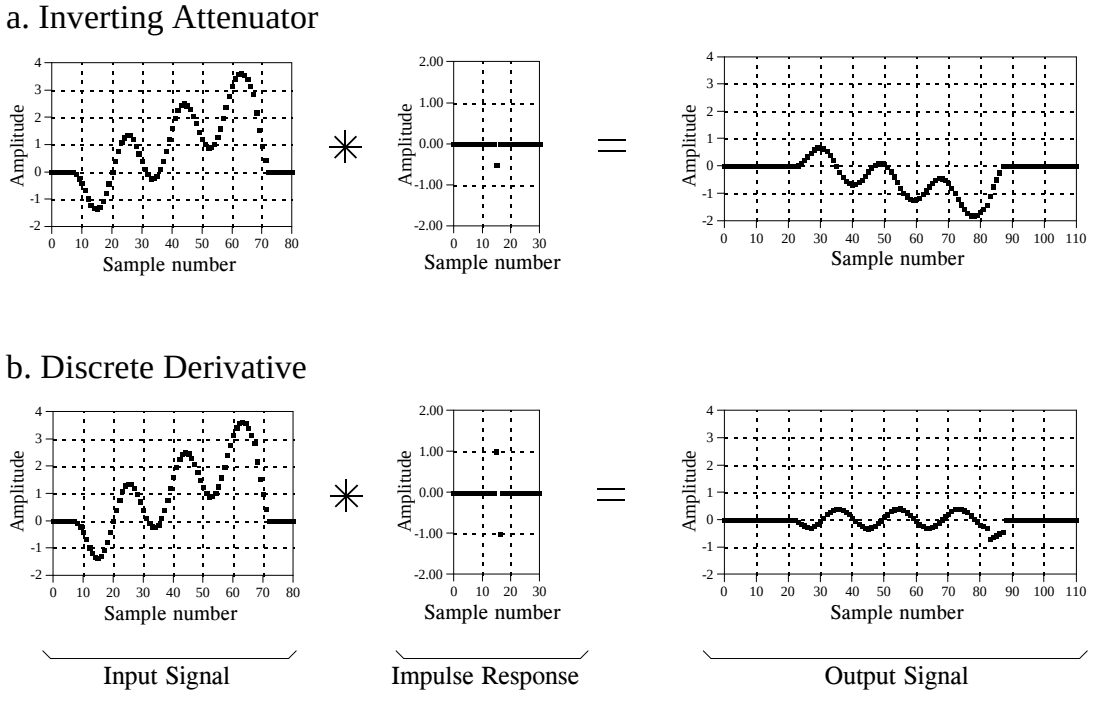


FIGURE 6-4
Examples of signals being processed using convolution. Many signal processing tasks use very simple impulse responses. As shown in these examples, dramatic changes can be achieved with only a few nonzero points.

equal to the length of the input signal, plus the length of the impulse response, minus one. For the signals in Figs. 6-3 and 6-4, each output signal is: 81 + 31 – 1 = 111 samples long. The input signal runs from sample 0 to 80, the impulse response from sample 0 to 30, and the output signal from sample 0 to 110.

Now we come to the detailed mathematics of convolution. As used in Digital Signal Processing, convolution can be understood in two separate ways. The first looks at convolution from the **viewpoint of the input signal**. This involves analyzing how each sample in the input signal *contributes* to many points in the output signal. The second way looks at convolution from the **viewpoint of the output signal**. This examines how each sample in the output signal has *received* information from many points in the input signal.

Keep in mind that these two perspectives are different ways of thinking about the same mathematical operation. The first viewpoint is important because it provides a *conceptual* understanding of how convolution pertains to DSP. The second viewpoint describes the *mathematics* of convolution. This typifies one of the most difficult tasks you will encounter in DSP: making your conceptual understanding fit with the jumble of mathematics used to communicate the ideas.

The Input Side Algorithm

Figure 6-5 shows a simple convolution problem: a 9 point input signal, $x[n]$, is passed through a system with a 4 point impulse response, $h[n]$, resulting in a $9 + 4 - 1 = 12$ point output signal, $y[n]$. In mathematical terms, $x[n]$ is convolved with $h[n]$ to produce $y[n]$. This first viewpoint of convolution is based on the fundamental concept of DSP: decompose the input, pass the components through the system, and synthesize the output. In this example, each of the nine samples in the input signal will contribute a scaled and shifted version of the impulse response to the output signal. These nine signals are shown in Fig. 6-6. Adding these nine signals produces the output signal, $y[n]$.

Let's look at several of these nine signals in detail. We will start with sample number four in the input signal, i.e., $x[4]$. This sample is at index number four, and has a value of 1.4. When the signal is decomposed, this turns into an impulse represented as: $1.4\,\delta[n-4]$. After passing through the system, the resulting output component will be: $1.4\,h[n-4]$. This signal is shown in the center box of the nine signals in Fig. 6-6. Notice that this is the impulse response, $h[n]$, multiplied by 1.4, and shifted four samples to the right. Zeros have been added at samples 0-3 and at samples 8-11 to serve as place holders. To make this more clear, Fig. 6-6 uses *squares* to represent the data points that come from the shifted and scaled impulse response, and *diamonds* for the added zeros.

Now examine sample $x[8]$, the last point in the input signal. This sample is at index number eight, and has a value of -0.5. As shown in the lower-right graph of Fig. 6-6, $x[8]$ results in an impulse response that has been shifted to the right by eight points and multiplied by -0.5. Place holding zeros have been added at points 0-7. Lastly, examine the effect of points $x[0]$ and $x[7]$. Both these samples have a value of zero, and therefore produce output components consisting of all zeros.

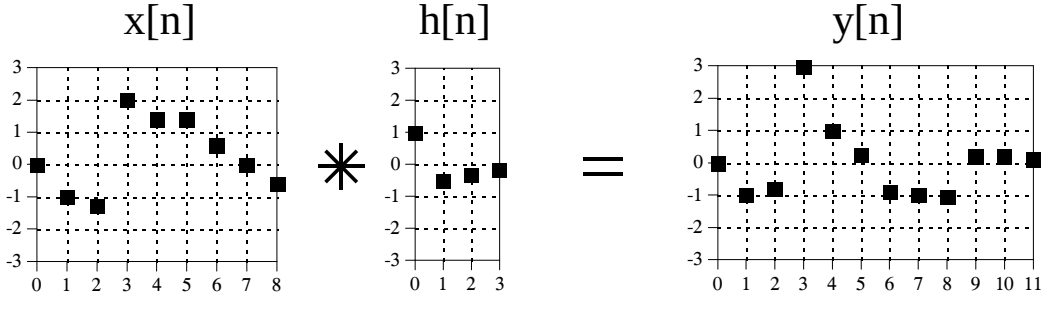


FIGURE 6-5
Example convolution problem. A nine point input signal, convolved with a four point impulse response, results in a twelve point output signal. Each point in the input signal contributes a scaled and shifted impulse response to the output signal. These nine scaled and shifted impulse responses are shown in Fig. 6-6.

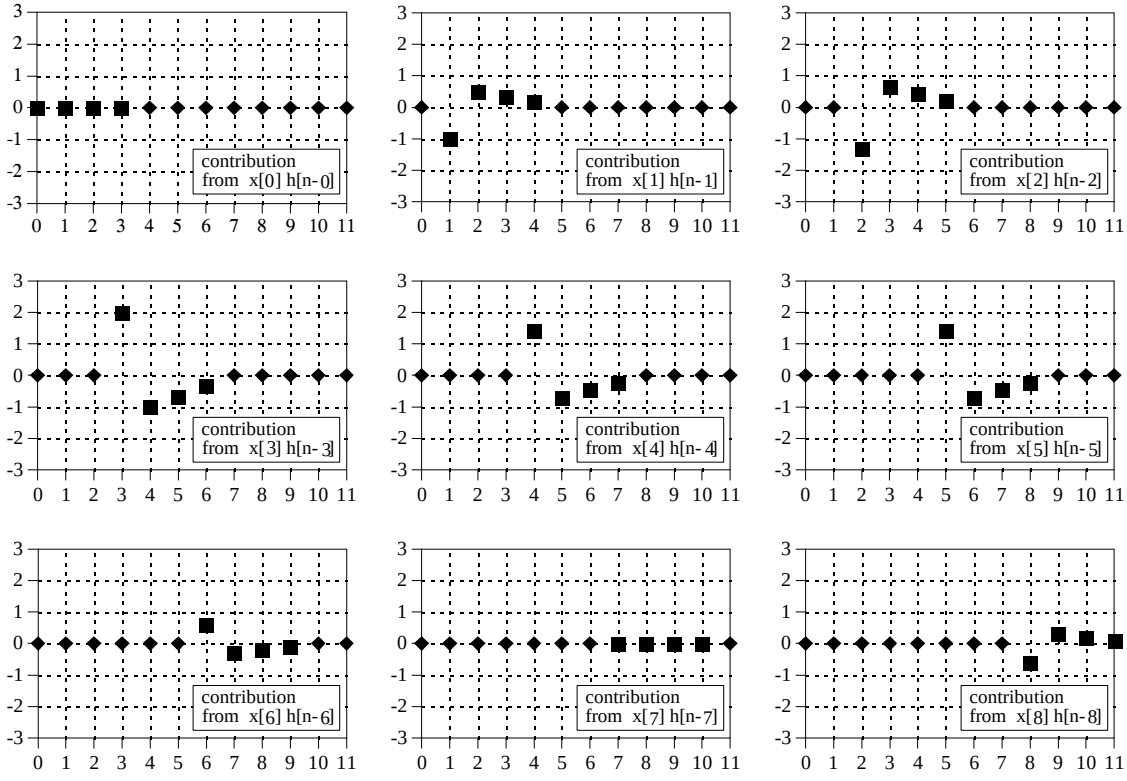


FIGURE 6-6
Output signal components for the convolution in Fig. 6-5. In these signals, each point that results from a scaled and shifted impulse response is represented by a square marker. The remaining data points, represented by diamonds, are zeros that have been added as place holders.

In this example, $x[n]$ is a nine point signal and $h[n]$ is a four point signal. In our next example, shown in Fig. 6-7, we will reverse the situation by making $x[n]$ a four point signal, and $h[n]$ a nine point signal. The same two waveforms are used, they are just swapped. As shown by the output signal components, the four samples in $x[n]$ result in four shifted and scaled versions of the nine point impulse response. Just as before, leading and trailing zeros are added as place holders.

But wait just one moment! The output signal in Fig. 6-7 is *identical* to the output signal in Fig. 6-5. This isn't a mistake, but an important property. Convolution is *commutative*: $a[n] * b[n] = b[n] * a[n]$. The mathematics does not care which is the input signal and which is the impulse response, only that *two signals are convolved with each other*. Although the mathematics may allow it, exchanging the two signals has no physical meaning in system theory. The input signal and impulse response are two totally different things and exchanging them doesn't make sense. What the commutative property provides is a *mathematical tool* for manipulating equations to achieve various results.

A program for calculating convolutions using the input side algorithm is shown in Table 6-1. Remember, the programs in this book are meant to convey *algorithms* in the simplest form, even at the expense of good programming style. For instance, all of the input and output is handled in **mythical** subroutines (lines 160 and 280), meaning we do not define how these operations are conducted. Do not skip over these programs; they are a key part of the material and you need to understand them in detail.

The program convolves an 81 point input signal, held in array X[ ], with a 31 point impulse response, held in array H[ ], resulting in a 111 point output signal, held in array Y[ ]. These are the same lengths shown in Figs. 6-3 and 6-4. Notice that the names of these arrays use upper case letters. This is a violation of the naming conventions previously discussed, because upper case letters are reserved for frequency domain signals. Unfortunately, the simple BASIC used in this book does not allow lower case variable names. Also notice that line 240 uses a star for *multiplication*. Remember, a star in a program means multiplication, while a star in an equation means convolution. A star in text (such as documentation or program comments) can mean either.

The mythical subroutine in line 160 places the input signal into X[ ] and the impulse response into H[ ]. Lines 180-200 set all of the values in Y[ ] to zero. This is necessary because Y[ ] is used as an accumulator to sum the output components as they are calculated. Lines 220 to 260 are the heart of the program. The FOR statement in line 220 controls a loop that steps through each point in the input signal, X[ ]. For each sample in the input signal, an inner loop (lines 230-250) calculates a scaled and shifted version of the impulse response, and adds it to the array accumulating the output signal, Y[ ]. This nested loop structure (one loop within another loop) is a key characteristic of convolution programs; become familiar with it.

```
100  'CONVOLUTION USING THE INPUT SIDE ALGORITHM
110                                '
120 DIM X[80]                      'The input signal, 81 points
130 DIM H[30]                      'The impulse response, 31 points
140 DIM Y[110]                     'The output signal, 111 points
150                                '
160 GOSUB XXXX                     'Mythical subroutine to load X[ ] and H[ ]
170                                '
180 FOR I% = 0 TO 110              'Zero the output array
190   Y(I%) = 0
200 NEXT I%
210                                '
220 FOR I% = 0 TO 80               'Loop for each point in X[ ]
230   FOR J% = 0 TO 30             'Loop for each point in H[ ]
240     Y[I%+J%] = Y[I%+J%] + X[I%]*H[J%]
250   NEXT J%
260 NEXT I%                        '(remember, * is multiplication in programs!)
270                                '
280 GOSUB XXXX                     'Mythical subroutine to store Y[ ]
290                                '
300 END
```

TABLE 6-1

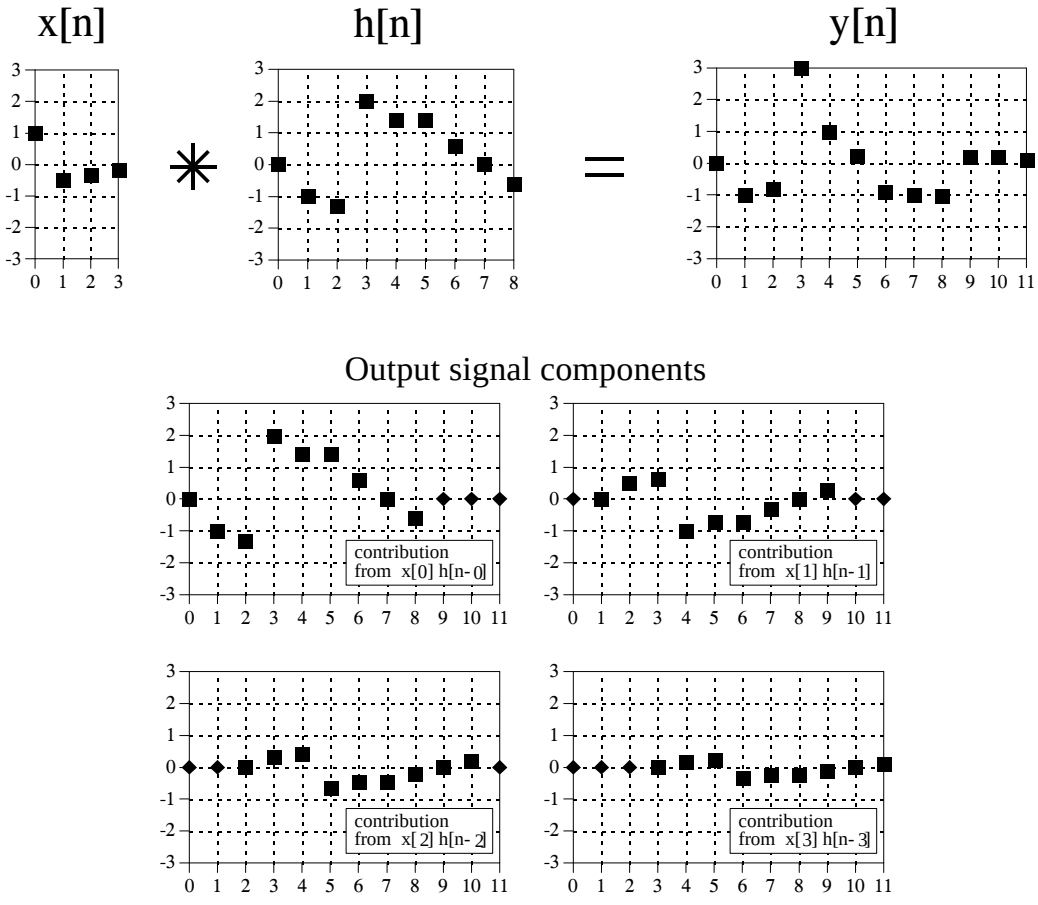


FIGURE 6-7
A second example of convolution. The waveforms for the input signal and impulse response are exchanged from the example of Fig. 6-5. Since convolution is commutative, the output signals for the two examples are identical.

Keeping the indexing straight in line 240 can drive you crazy! Let's say we are halfway through the execution of this program, so that we have just begun action on sample X[40], i.e., I% = 40. The inner loop runs through each point in the impulse response doing three things. First, the impulse response is *scaled* by multiplying it by the value of the input sample. If this were the only action taken by the inner loop, line 240 could be written, Y[J%] = X[40]✳H[J%]. Second, the scaled impulse is *shifted* 40 samples to the right by adding this number to the index used in the output signal. This second action would change line 240 to: Y[40+J%] = X[40]✳H[J%]. Third, Y[ ] must accumulate (*synthesize*) all the signals resulting from each sample in the input signal. Therefore, the new information must be added to the information that is already in the array. This results in the final command: Y[40+J%] = Y[40+J%] + X[40]✳H[J%]. Study this carefully; it is *very* confusing, but *very* important.

The Output Side Algorithm

The first viewpoint of convolution analyzes how each sample in the *input signal* affects many samples in the output signal. In this second viewpoint, we reverse this by looking at individual samples in the *output signal*, and finding the contributing points from the input. This is important from both mathematical and practical standpoints. Suppose that we are given some input signal and impulse response, and want to find the convolution of the two. The most straightforward method would be to write a program that loops through the *output signal*, calculating one sample on each loop cycle. Likewise, equations are written in the form: $y[n]$ = *some combination of other variables*. That is, sample n in the output signal is equal to some combination of the many values in the input signal and impulse response. This requires a knowledge of how each sample in the output signal can be calculated independently of all other samples in the output signal. The output side algorithm provides this information.

Let's look at an example of how a single point in the output signal is influenced by several points from the input. The example point we will use is $y[6]$ in Fig. 6-5. This point is equal to the sum of all the sixth points in the nine output components, shown in Fig. 6-6. Now, look closely at these nine output components and identify which can affect $y[6]$. That is, find which of these nine signals contains a nonzero sample at the sixth position. Five of the output components only have *added* zeros (the diamond markers) at the sixth sample, and can therefore be ignored. Only four of the output components are capable of having a nonzero value in the sixth position. These are the output components generated from the input samples: $x[3]$, $x[4]$, $x[5]$, and $x[6]$. By adding the sixth sample from each of these output components, $y[6]$ is determined as: $y[6] = x[3]h[3] + x[4]h[2] + x[5]h[1] + x[6]h[0]$. That is, four samples from the input signal are multiplied by the four samples in the impulse response, and the products added.

Figure 6-8 illustrates the output side algorithm as a **convolution machine**, a flow diagram of how convolution occurs. Think of the input signal, $x[n]$, and the output signal, $y[n]$, as fixed on the page. The convolution machine, everything inside the dashed box, is free to move left and right as needed. The convolution machine is positioned so that its output is aligned with the output sample being calculated. Four samples from the input signal fall into the inputs of the convolution machine. These values are multiplied by the indicated samples in the impulse response, and the products are added. This produces the value for the output signal, which drops into its proper place. For example, $y[6]$ is shown being calculated from the four input samples: $x[3]$, $x[4]$, $x[5]$, and $x[6]$.

To calculate $y[7]$, the convolution machine moves one sample to the right. This results in another four samples entering the machine, $x[4]$ through $x[7]$, and the value for $y[7]$ dropping into the proper place. This process is repeated for all points in the output signal needing to be calculated.

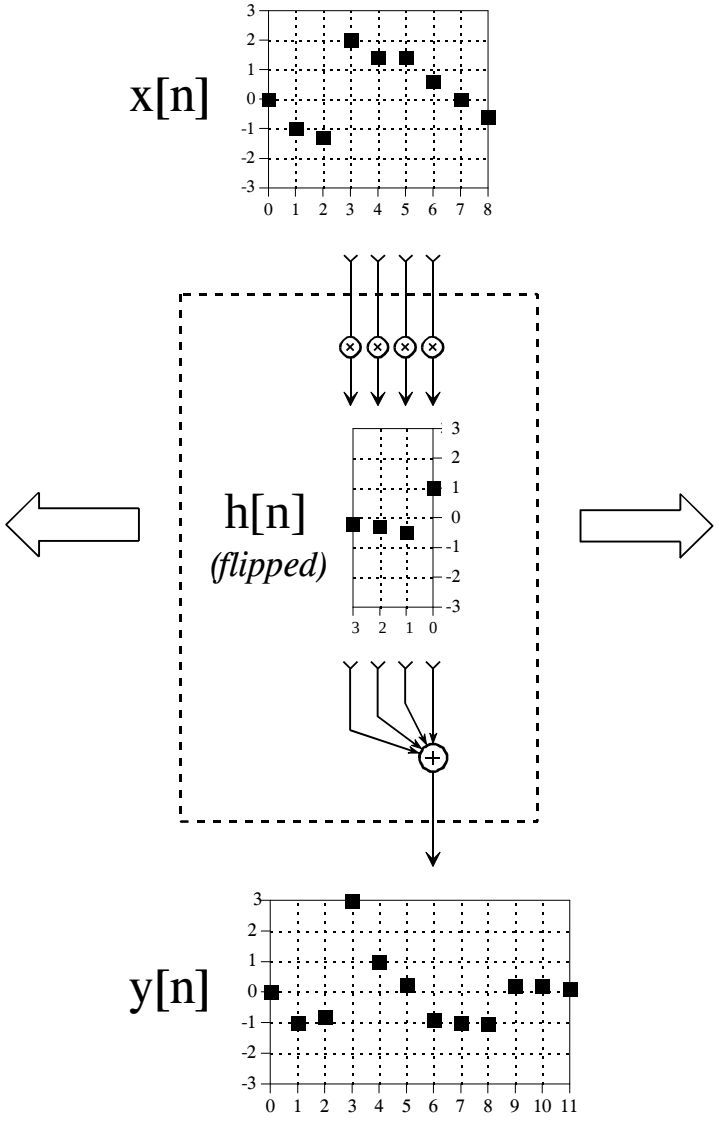


FIGURE 6-8
The convolution machine. This is a flow diagram showing how each sample in the output signal is influenced by the input signal and impulse response. See the text for details.

The arrangement of the impulse response *inside* the convolution machine is very important. The impulse response is *flipped left-for-right*. This places sample number zero on the right, and increasingly positive sample numbers running to the left. Compare this to the normal impulse response in Fig. 6-5 to understand the geometry of this flip. Why is this flip needed? It simply falls out of the mathematics. The impulse response describes how each point in the input signal affects the output signal. This results in each point in the output signal being affected by points in the input signal weighted by a *flipped* impulse response.

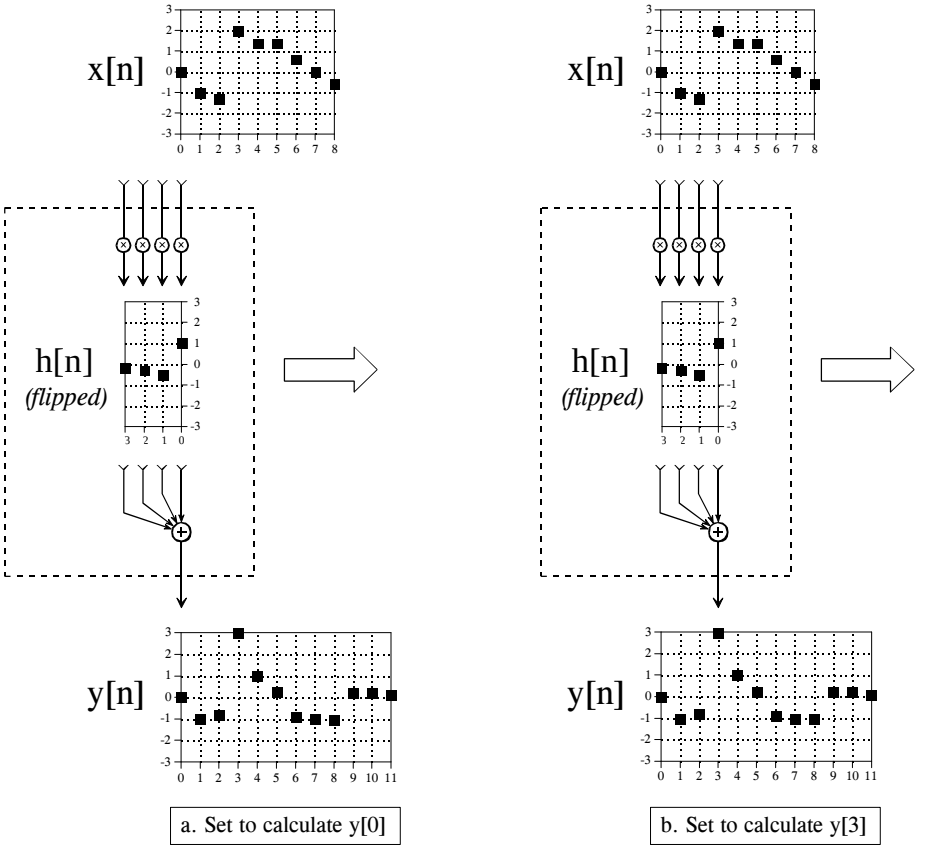


FIGURE 6-9
The convolution machine in action. Figures (a) through (d) show the convolution machine set to calculate four different output signal samples, y[0], y[3], y[8], and y[11].

Figure 6-9 shows the convolution machine being used to calculate several samples in the output signal. This diagram also illustrates a real nuisance in convolution. In (a), the convolution machine is located fully to the left with its output aimed at $y[0]$. In this position, it is trying to receive input from samples: $x[-3]$, $x[-2]$, $x[-1]$, and $x[0]$. The problem is, three of these samples: $x[-3]$, $x[-2]$, and $x[-1]$, do not exist! This same dilemma arises in (d), where the convolution machine tries to accept samples to the right of the defined input signal, points $x[9]$, $x[10]$, and $x[11]$.

One way to handle this problem is by *inventing* the nonexistent samples. This involves adding samples to the ends of the input signal, with each of the added samples having a value of *zero*. This is called **padding** the signal with zeros. Instead of trying to access a nonexistent value, the convolution machine receives a sample that has a value of zero. Since this zero is eliminated during the multiplication, the result is mathematically the same as *ignoring* the nonexistent inputs.

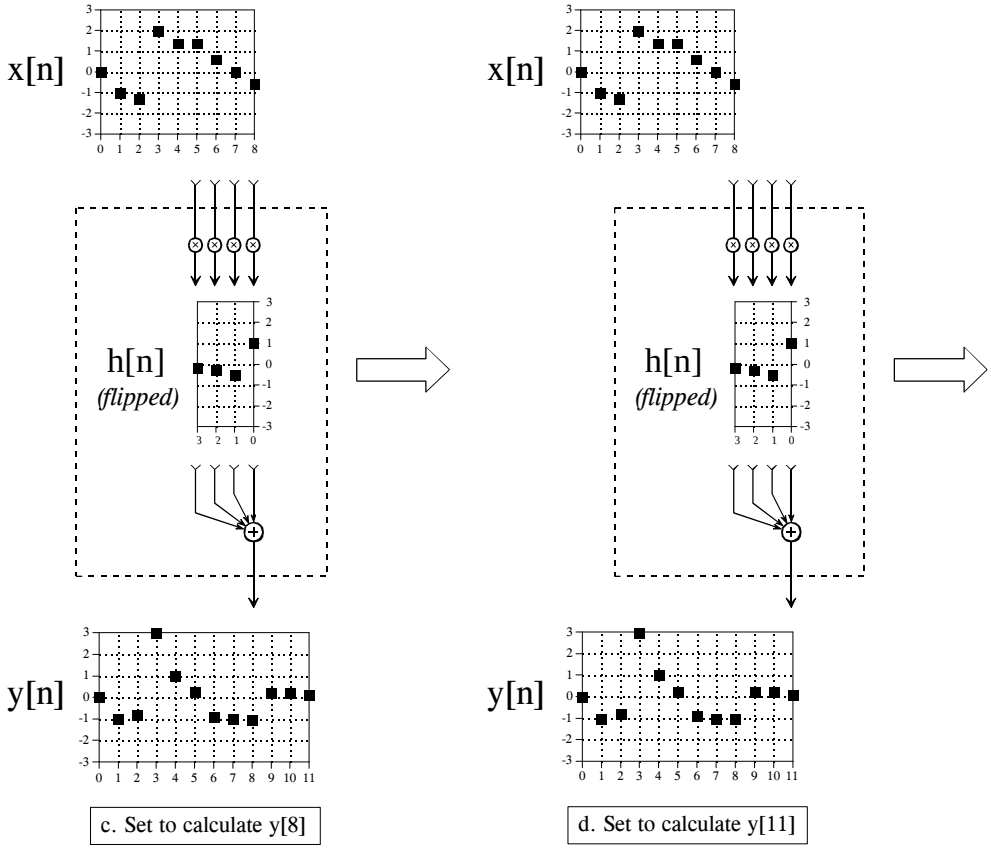


Figure 6-9 (continued)

The important part is that the far left and far right samples in the output signal are based on *incomplete* information. In DSP jargon, **the impulse response is not fully immersed in the input signal**. If the impulse response is M points in length, the first *and* last $M-1$ samples in the output signal are based on less information than the samples between. This is analogous to an electronic circuit requiring a certain amount of time to stabilize after the power is applied. The difference is that this transient is easy to ignore in electronics, but very prominent in DSP.

Figure 6-10 shows an example of the trouble these end effects can cause. The input signal is a sine wave plus a DC component. The desire is to remove the DC part of the signal, while leaving the sine wave intact. This calls for a high-pass filter, such as the impulse response shown in the figure. The problem is, the first and last 30 points are a mess! The shape of these end regions can be understood by imagining the input signal padded with 30 zeros on the left side, samples $x[-1]$ through $x[-30]$, and 30 zeros on the right, samples $x[81]$ through $x[110]$. The output signal can then be viewed as a filtered version of this longer waveform. These "end effect" problems are widespread in

DSP. As a general rule, expect that the beginning and ending samples in processed signals will be quite useless.

Now the math. Using the convolution machine as a guideline, we can write the standard equation for convolution. If $x[n]$ is an N point signal running from 0 to N-1, and $h[n]$ is an M point signal running from 0 to M-1, the convolution of the two: $y[n] = x[n] * h[n]$, is an $N+M$-1 point signal running from 0 to $N+M$-2, given by:

EQUATION 6-1
The convolution summation. This is the formal definition of convolution, written in the shorthand: $y[n] = x[n] * h[n]$. In this equation, $h[n]$ is an M point signal with indexes running from 0 to M-1.

$$y[i] = \sum_{j=0}^{M-1} h[j]\, x[i-j]$$

This equation is called the **convolution sum**. It allows each point in the output signal to be calculated independently of all other points in the output signal. The index, i, determines which sample in the output signal is being calculated, and therefore corresponds to the left-right position of the convolution machine. In computer programs performing convolution, a loop makes this index run through each sample in the output signal. To calculate one of the output samples, the index, j, is used *inside* of the convolution machine. As j runs through 0 to M-1, each sample in the impulse response, $h[j]$, is multiplied by the proper sample from the input signal, $x[i-j]$. All these products are added to produce the output sample being calculated. Study Eq. 6-1 until you fully understand how it is implemented by the convolution machine. Much of DSP is based on this equation. (Don't be confused by the n in $y[n] = x[n] * h[n]$. This is merely a place holder to indicate that *some* variable is the index into the array. Sometimes the equations are written: $y[\,] = x[\,] * h[\,]$, just to avoid having to bring in a meaningless symbol).

Table 6-2 shows a program for performing convolutions using the *output side algorithm*, a direct use of Eq. 6-1. This program produces the same output signal as the program for the *input side algorithm*, shown previously in Table 6-1. Notice the main difference between these two programs: the input side algorithm loops through each sample in the *input signal* (line 220 of Table 6-1), while the output side algorithm loops through each sample in the *output signal* (line 180 of Table 6-2).

Here is a detailed operation of this program. The FOR-NEXT loop in lines 180 to 250 steps through each sample in the output signal, using I% as the index. For each of these values, an inner loop, composed of lines 200 to 230, calculates the value of the output sample, Y[I%]. The value of Y[I%] is set to zero in line 190, allowing it to accumulate the products inside of the convolution machine. The FOR-NEXT loop in lines 200 to 240 provide a direct implementation of Eq. 6-1. The index, J%, steps through each

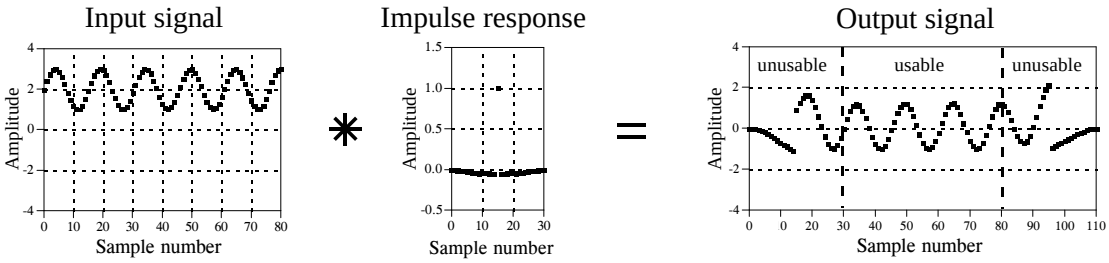


FIGURE 6-10
End effects in convolution. When an input signal is convolved with an *M* point impulse response, the first and last *M*-1 points in the output signal may not be usable. In this example, the impulse response is a high-pass filter used to remove the DC component from the input signal.

sample in the impulse response. Line 230 provides the multiplication of each sample in the impulse response, H[J%], with the appropriate sample from the input signal, X[I%-J%], and adds the result to the accumulator.

In line 230, the sample taken from the input signal is: X[I%-J%]. Lines 210 and 220 prevent this from being outside the defined array, X[0] to X[80]. In other words, this program handles undefined samples in the input signal by *ignoring* them. Another alternative would be to define the input signal's array from X[-30] to X[110], allowing 30 zeros to be padded on each side of the true data. As a third alternative, the FOR-NEXT loop in line 180 could be changed to run from 30 to 80, rather than 0 to 110. That is, the program would only calculate the samples in the output signal where the impulse response is *fully immersed* in the input signal. The important thing is that you must use one of these three techniques. If you don't, the program will crash when it tries to read the out-of-bounds data.

```
100  'CONVOLUTION USING THE OUTPUT SIDE ALGORITHM
110                              '
120 DIM X[80]                    'The input signal, 81 points
130 DIM H[30]                    'The impulse response, 31 points
140 DIM Y[110]                   'The output signal, 111 points
150                              '
160 GOSUB XXXX                   'Mythical subroutine to load X[ ] and H[ ]
170                              '
180 FOR I% = 0 TO 110            'Loop for each point in Y[ ]
190   Y[I%] = 0                  'Zero the sample in the output array
200   FOR J% = 0 TO 30           'Loop for each point in H[ ]
210     IF (I%-J% < 0)           THEN GOTO 240
220     IF (I%-J% > 80)          THEN GOTO 240
230     Y(I%) = Y(I%) + H(J%) * X(I%-J%)
240   NEXT J%
250 NEXT I%
260                              '
270 GOSUB XXXX                   'Mythical subroutine to store Y[ ]
280                              '
290 END
```

TABLE 6-2

The Sum of Weighted Inputs

The characteristics of a linear system are completely described by its impulse response. This is the basis of the input side algorithm: each point in the input signal contributes a scaled and shifted version of the impulse response to the output signal. The mathematical consequences of this lead to the output side algorithm: each point in the output signal receives a contribution from many points in the input signal, multiplied by a *flipped* impulse response. While this is all true, it doesn't provide the full story on why convolution is important in signal processing.

Look back at the convolution machine in Fig. 6-8, and ignore that the signal inside the dotted box is an *impulse response*. Think of it as a set of **weighing coefficients** that happen to be embedded in the flow diagram. In this view, each sample in the output signal is equal to a *sum of weighted inputs*. Each sample in the output is influenced by a region of samples in the input signal, as determined by what the weighing coefficients are chosen to be. For example, imagine there are ten weighing coefficients, each with a value of one-tenth. This makes each sample in the output signal the *average* of ten samples from the input.

Taking this further, the weighing coefficients do not need to be restricted to the *left side* of the output sample being calculated. For instance, Fig. 6-8 shows $y[6]$ being calculated from: $x[3]$, $x[4]$, $x[5]$, and $x[6]$. Viewing the convolution machine as a sum of weighted inputs, the weighing coefficients could be chosen *symmetrically* around the output sample. For example, $y[6]$ might receive contributions from: $x[4]$, $x[5]$, $x[6]$, $x[7]$, and $x[8]$. Using the same indexing notation as in Fig. 6-8, the weighing coefficients for these five inputs would be held in: $h[2]$, $h[1]$, $h[0]$, $h[-1]$, and $h[-2]$. In other words, the impulse response that corresponds to our selection of symmetrical weighing coefficients requires the use of *negative indexes*. We will return to this in the next chapter.

Mathematically, there is only one concept here: convolution as defined by Eq. 6-1. However, science and engineering problems approach this single concept from two distinct directions. Sometimes you will want to think of a system in terms of what its impulse response looks like. Other times you will understand the system as a set of weighing coefficients. You need to become familiar with both views, and how to toggle between them.