CHAPTER

5

Linear Systems Analysis in the Time Domain—Convolution

5.1 GOALS OF THIS CHAPTER

You now have expertise in the fundamentals of signal analysis, including basic and advanced time-domain measurements (mean, root mean square, standard deviation, variance, and correlations, including auto- and cross-correlation) and frequency-domain analysis involving time–frequency transformations with emphasis on signal spectrum. Now it is time to move on to systems: processes that moderate, influence, and/or produce signals.

Ultimately, we want to know how a given physiological system does what it does, but we start with a more modest goal: to have a definitive understanding of what the system does. So the primary objective of "linear system analysis" is to determine how a system responds to any input, no matter how complicated the system or the input. This input/output determination can be done in either the time domain or frequency domain. In both approaches, we start with a definition of the system itself. These representations are quite different in the two domains, but they achieve the same end: a mathematical function that allows us to predict the output of the system to any input. In this chapter, we look at the time domain representation of a system, whereas Chapter 6 takes the frequency-domain approach.

Specific topics include:

- Basic goals of linear signal analysis, system constraints, superposition
- Defining a system in the time domain: the impulse function (i.e., signal) and impulse response
- Finding a system's output to any specific input signal in the time domain: convolution

5.2 LINEAR SYSTEMS ANALYSIS—AN OVERVIEW

Systems act on signals. The objective of systems analysis is to describe how a system modifies a signal: any (linear) system and any signal(s). To be able to predict the behavior of complex processes in response to complex stimuli, we usually impose rather severe

Circuits, Signals, and Systems for Bioengineers
https://doi.org/10.1016/B978-0-12-809395-5.00005-9

simplifications and/or assumptions. The most common assumptions are: (1) that the process responds linearly to all inputs, and (2) that the basic characteristics do not change over time. In Section 1.4 we note that these two assumptions define an LTI (linear, time invariant) system. These assumptions allow us to apply a powerful array of mathematical tools known as linear systems analysis.

Of course, living systems change over time, they are adaptive and often nonlinear. But the power of linear systems analysis is sufficiently seductive that assumptions or approximations are made so that these tools can be used. Linearity can be approximated by using small-signal conditions since, when a system's range of action is restricted, it often behaves more or less linearly. Alternatively, piecewise linear approaches can be used where the analysis is confined to operating ranges over which the system behaves linearly. In Chapter 9, we learn computer simulation techniques that allow us to analyze systems with certain types of nonlinearity. To deal with processes that change over time, we can use the same approach underlying the short-term Fourier transform: restrict the analysis time frame to a period when the system can be considered time invariant.

5.2.1 Superposition and Linearity

Finding the output of any system to any input is a tall order and the only way we can achieve it is to break up the signal. For the time-domain analysis described here, we break up the signal into little slices of time. In frequency-domain analyses as described in the next chapter, we break up the signal into frequency components (i.e., sinusoids). Either way, we determine how the system responds to each of these components as if it was acting alone. To get the output signal, we put those individual output components back together. For this divide-and-conquer approach, we rely on an important concept known as superposition; for superposition we need linearity.

Superposition states that when multiple influences act on some process, the resultant behavior is the sum of the process's response to each influence acting alone. This means that when a system receives two or more signals, a valid solution can be obtained by solving for the response to each signal in isolation, and then algebraically summing these partial solutions. The sources could be anywhere in the system or even from different locations. This is exactly what we need for our decomposition technique to work.

The superposition principle is a consequence of linearity and applies only to LTI systems. It makes all signal processing tools more powerful since it enables decomposition strategies, either in the frequency or the time domain. Examples using the principle of superposition are found throughout this book.

Two time-domain approaches can be used to find the output of an LTI system to almost any input. Both decompose the signal into slices of time. Slicing the input signal into small, or even infinitesimal, segments uses the approach of integral calculus where a function is divided up into infinitely small segments, the solution to each segment is determined, and these solutions are summed. This approach is termed "convolution," which applies this time-slice trick to the entire system in one operation. For continuous signals, the signal segments can be infinitesimal and integration used to sum up the contributions of each segment. If digital signals are involved, then the minimum segment size is one sample (i.e., one data point) so the size of the time slice depends on the sample interval.

The second time-domain approach is a very powerful computer-based method called "simulation" and is described in Chapter 9. In this inherently digital approach, the response of each element to the first time slice is found element by element. This process is repeated for each subsequent time slice and including the last time slice in the input signal. It is as computationally intensive as it sounds, but it gives access to internal signals as well as the output signal and can be used on systems that contain nonlinearities.

5.3 A SLICE IN TIME: THE IMPULSE SIGNAL

The convolution approach takes advantage of the fact that any linear system responds in a characteristic way to sufficiently short time[1] slices, or pulse signals, irrespective of pulse amplitude. Pulse amplitude simply scales this characteristic response up or down, Figure 5.1. A pulse signal having the smallest possible time slice is called the "impulse function," or "impulse signal," and the system's response to this signal is called the "impulse response."

Since the shape of the impulse response is unique to a given system, it can be used as the time-domain representation of the system. For a unique representation, we need to agree on standard impulse amplitude. As we show, it is better to define a standard impulse function area as having a given value rather than a given amplitude. By convention, we set the area of a standard impulse to 1.0. For a digital impulse signal this is no problem: the area is the amplitude multiplied by one sample interval, T_s, so the amplitude should be $1/T_s$ to produce an area of 1.0. For a continuous signal, the width of an impulse function, Δt, becomes

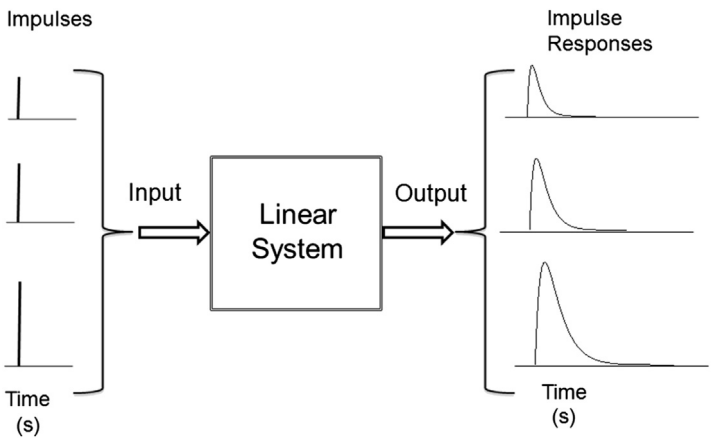


FIGURE 5.1 A sufficiently short input signal[1] is called the impulse function. For all linear systems, the response to an impulse has a characteristic shape determined by the system. This shape is the same regardless of the amplitude of the impulse. This characteristic response is called the impulse response and essentially defines the system in the time domain.

[1]By sufficiently short, we mean infinitely short for continuous systems, and one sample interval (i.e., T_s) for discrete systems. For real-world systems, we can determine the maximum slice size empirically as shown in Example 5.1.

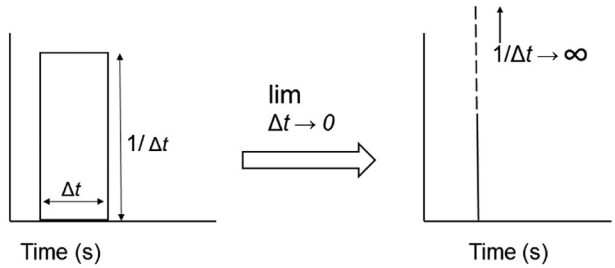


FIGURE 5.2 A continuous impulse function is a theoretical construct that has a width that goes to zero, an amplitude that goes to infinity, but an area of 1.0. This function is called a delta(δ)-function.

infinitely small, $\Delta t \rightarrow 0$, so the amplitude, $1/\Delta t$ must become infinite as $\Delta t \rightarrow 0$, at least in theory, to maintain an area of 1.0, Figure 5.2. (The need for infinite amplitude to keep the area of an infinitely short pulse equal to 1.0 is the reason we define area rather than amplitude.) This theoretical impulse input is called the "delta function" and is notated $\delta(t)$.

5.3.1 Real-World Impulse Signals

A signal that is infinitely short with infinite amplitude is not compatible with real-world situations. If we want to generate a real impulse function, we use a pulse that is sufficiently short and has enough amplitude to produce a reasonable response signal. (In real-world situations there is always noise present and we would like the impulse response to be much larger than the noise.) How short is sufficiently short? This depends on the dynamics of the system: it should be much shorter than the fastest system response. This may still sound vague, but there is a simple way to set the maximum limit of a practical impulse input empirically as shown in our first example.

EXAMPLE 5.1

An unknown system is represented by the MATLAB routine:

`y = unknown_sys5_1(x).m` where `x` is the input to the system and `y` is its output. Although we do not know any details of the system, we suspect that it responds to stimulus changes in around a tenth of a second (0.1 s). Find the maximum pulse width that can be considered sufficiently short to be taken as an impulse input to this system.

Solution: An impulse should produce a characteristic response from any given system. If we make it shorter the response amplitude will decrease, but the shape will stay the same. So if we compare responses to two pulse signals, one shorter than the other, and they both give the same-shaped responses, then both can be considered short enough to be impulse functions. Since we know the system in question responds on the order of 0.1 s, we compare the system's response to pulses having widths ranging from 100 ms down to 2.0 ms. Specifically, we compare the following pairs of pulse widths: 50 and 100 ms; 10 and 25 ms; 5.0 and 10 ms; and 2.0 and 5.0 ms. These are just guesses, so we write the MATLAB code so that pulse width can be easily changed. To make the comparison easier, we normalize the output responses to a maximum value of 1.0.

We need to set a sample interval and data length. To get a 2.0-ms pulse, we need to have a sample frequency of at least 500 Hz, so we select f_s to be 1000 to be conservative. As to data length,

given responses take around 0.1 s we start with a data length of 0.2 s. Again we can always make the time frame shorter or longer if need be.

```
% Example 5.1  Example to evaluate the responses of an unknown
% system to pulses of various widths.
%
PW = [50 10 5 5; 100 25 10 2];              % Pulse widths in msec
fs = 1000;                                  % Sample frequency
N = round(.2 *fs);                          % Data length for 0.2 sec
t = (0:N-1)/fs;                             % Time vector for plotting
%
for k = 1:4                                 % Do four comparison plots
  subplot(2,2,k); hold on;                  % Plot two by two
  x = [ones(1,PW(1,k)) zeros(1,N-PW(1,k))]; % Generate pulse signal
  y = unknown_sys5_1(x);                    % Stimulate system, get response
  y = y/max(y);                             % Normalize peak to 1.0
  plot(t,y,'k'); hold on;                   % Plot pulse response
  x = [ones(1,PW(2,k)) zeros(1,N-PW(2,k))]; % Generate pulse signal
  y = unknown_sys5_1(x);                    % Stimulate system, get response
  y = y/max(y);                             % Normalize peak to 1.0
  plot(t,y,'k');                            % Plot pulse response
       .......labels and text.......
end
```

Results: The four comparison pulse responses are shown in Figure 5.3. All responses are scaled to a maximum amplitude of 1.0. (Since the different pulse widths have different energies, they produce

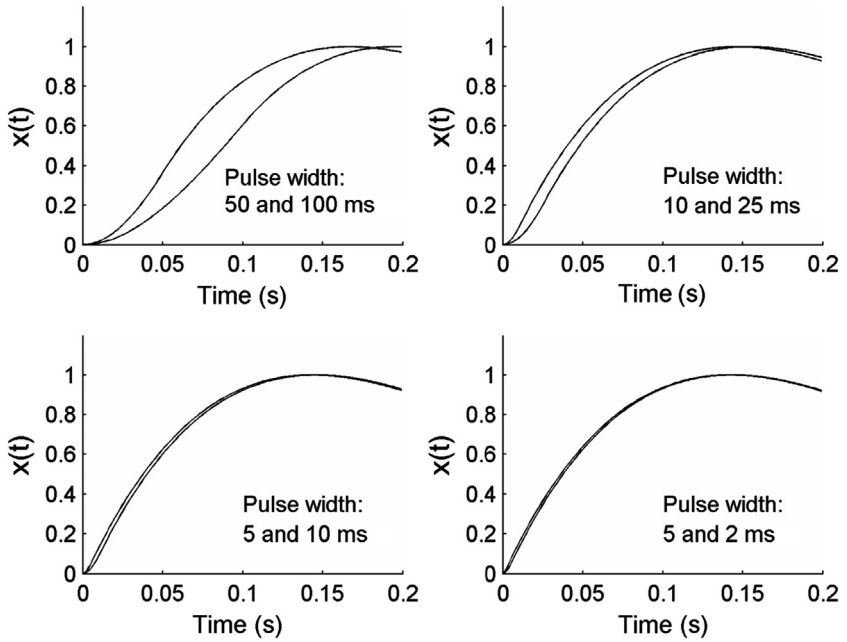


FIGURE 5.3 Pulse responses of an unknown system to different combinations of pulse width. These responses have been normalized to a maximizing value of 1.0. When the inputs have widths of either 2.0 or 5.0 ms, the normalized responses are nearly identical, indicating that both can be considered impulse signals with respect to this system.

different response amplitudes, irrespective of shape.) Figure 5.3 shows that as the pulses reduce in width, the responses become more similar despite differences in the pulse width of the input signals. There is very little difference between the responses generated by the 5- and 2.5-ms pulses, so that a 5-ms pulse (or less) is close enough to a true impulse for this system. This example is a realistic simulation of the kind of experiment one might do to determine the impulse response empirically, provided the physical system and some type of pulse stimulator are available. This also assumes the system can be stimulated with pulse waveforms and the response monitored, not always the case with biological systems.

5.3.2 The Impulse Signal in the Frequency Domain

The impulse signal has a very special frequency-domain representation. Again, we show this by example. In the next example, we find the magnitude spectra for two of the pulse signals used in Example 5.1 and the magnitude spectrum of a true discrete impulse signal: a signal that has a value of 1.0 for one the first sample and zero everywhere else.

EXAMPLE 5.2

Find the magnitude spectra of two pulse signals having widths of 5 and 2 ms and a true discrete impulse signal. The true impulse signal should have a value of 1.0 for the first sample and zeros everywhere else. Assume a sample rate of 1 kHz as in the last example, but make the signal length 1000 samples. Plot the spectra superimposed, but scale the maximum value of each magnitude spectrum to 1.0 for easy comparison. Label the three spectra. As always, plot only the valid spectral points and label the plots.

Solution: We can borrow the same code used in Example 5.1. Since $f_s = 1$ kHz, the pulse width vector has only three entries of 5, 2, and 1 representing the three pulse widths in milliseconds. Instead of using the pulse signals as inputs to an unknown system, we find their magnitude spectra using the `fft` routine. We normalize the magnitude spectra to have a maximum value of 1.0 then plot the three spectra superimposed.

```
% Example 5.2 Find the magnitude spectra of two pulses having a width of
%  5 and 2 msec and a true discrete impulse signal.
%
PW = [5, 2, 1];          % Pulse widths in msec
fs = 1000;               % Sample frequency (given)
N = 1000;                % Data length (given)
N2 = 500;                % Valid spectral points
f = (1:N)*fs/N;          % Frequency vector for plotting
%
for k = 1:3              % Do 3 spectral plots
  x = [ones(1,PW(k)) zeros(1,N-PW(k))];      % Generate pulse signal
  X = abs(fft(x));       % Find magnitude spectrum
  X = X/max(X);          % Normalize the spectra
  plot(f(1:N2),X(1:N2),'k'); hold on;        % Plot pulse spectra
  ……label spectral curves…….
end
```

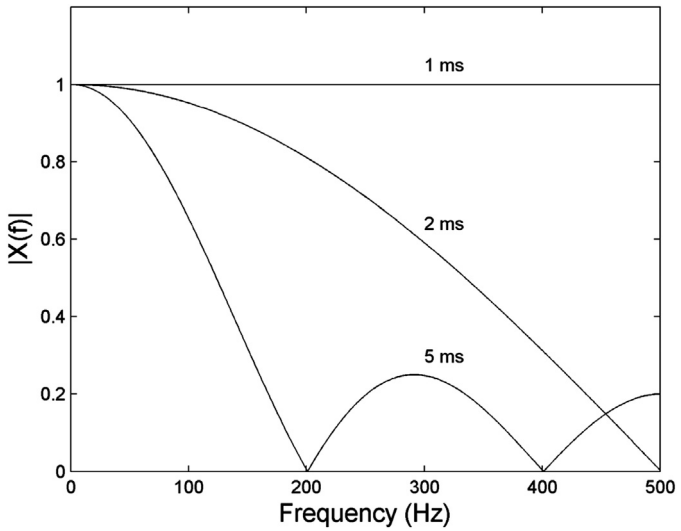


FIGURE 5.4 The magnitude spectra of three pulse signals. Since $f_s = 1$ kHz, the 1.0-ms pulse width signal is a true discrete impulse signal. Although the longer signals decrease in frequency (as $|\sin(x)/x|$, see Example 3.3), the impulse signal's spectrum is a constant over all valid frequencies, 0 to $f_s/2$.

Results: The three spectral curves are shown in Figure 5.4. As shown in Example 3.3, the magnitude spectrum of a pulse signal has a shape given by |sin(x)/x)| (see Figure 3.12). This is seen for the 2- and 5-ms pulses, where the shorter pulse produces a spectrum that falls off less rapidly with increasing frequency, but still goes to zero at the Nyquist frequency, $f_s/2$. The true impulse has a much different magnitude spectrum. It is a constant value across all frequencies between 0 and $f_s/2$ Hz. Its phase spectrum is also a constant. As shown in one of the problems, the phase angle is 0.0 degree over the frequency range of 0 to $f_s/2$ Hz.

The spectrum of the true impulse is quite different and rather remarkable. It contains an equal amount of energy for all the valid frequencies in the signal, a property that can be very useful in exploring the frequency characteristics of a system. Just as a signal can have a spectrum, so can a system.[2] A system's spectrum shows how that system attenuates or enhances an input signal as a function of frequency. The impulse response can be used to find a system's spectrum. Here is the rationale: if the input signal in the frequency domain is a constant across all frequencies, the output frequencies show how the system modifies signals as a function of frequency. In other words, if the impulse has a constant spectrum, the spectrum of the impulse response must be identical to the spectrum of the system. So to find the spectrum of a system, we only need to convert the impulse response to the frequency domain using the Fourier transform.

[2]The spectrum of a system is mathematically embodied in the "transfer function" as described extensively in the next chapter.

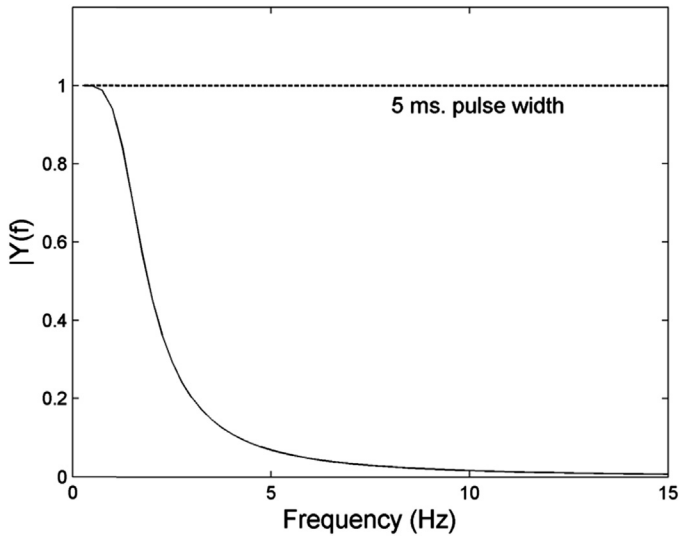


FIGURE 5.5 The magnitude spectrum of the unknown system from Example 5.1. The response of this system decreases rapidly for frequencies above 2.0 Hz. In the frequency domain it looks like a low-pass filter, although its actual function is unknown. The dashed line shows the magnitude spectrum of a 5.0-ms pulse signal, which appears to be constant over this limited frequency range. This explains the finding in Example 5.1 that such a pulse can serve as an impulse signal.

We can use the impulse signal to find the frequency characteristics of the unknown system used in Example 5.1. When the input is effectively an impulse, the spectrum of the output is shown in Figure 5.5 (solid line). (Note the expanded frequency scale ranges between 0 and 15 Hz.) The system is seen to decrease for frequencies that are above 2.0 Hz. This system, whatever its real purpose, acts like a low-pass filter. It can respond to signals having low-frequency energy, but for input signals much above 5 Hz there is little response. The dashed line in Figure 5.5 shows the magnitude spectrum of a 5.0-ms pulse to be almost constant over this limited frequency range. That is why it acts like an impulse in Example 5.1. For the limited range of frequencies to which the unknown system is capable of responding, the 5.0-ms pulse signal looks like an impulse.

This illustrates another way to determine whether a short pulse can be considered an impulse if you know the frequency characteristics of the system. Just compare the system's spectrum with that of the short pulse. If the pulse spectrum is more or less flat over the range of the system's nonzero spectrum, it can be considered an impulse. This idea is presented in one of the problems. In the next two chapters, we become deeply involved in the frequency characteristics of the system and we will return to the impulse input with its unique spectral properties.

5.4 USING THE IMPULSE RESPONSE TO FIND A SYSTEMS OUTPUT TO ANY INPUT—CONVOLUTION

Any signal can be chopped up into time slices; any signal can be represented as a sequence of impulse signals. Each pulse will produce its own impulse response. All these individual

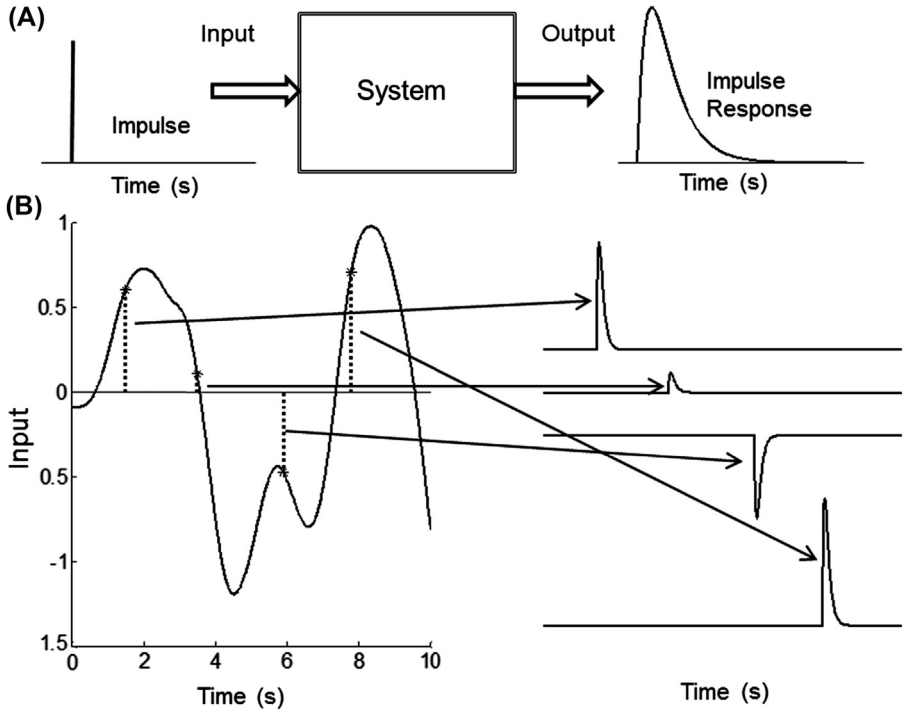


FIGURE 5.6 (A) An impulse input produces a characteristic response known as the impulse response. (B) A general signal can be represented as a number (possibly infinite) of impulses in sequence. Each impulse produces its own response and all responses have the same shape, a shape characteristic of the system. The only difference is the amplitude of the impulse response, which depends on the amplitude of the signal.

impulse responses have the same shape; the only difference is amplitude, which is scaled by the input signal's amplitude during a given time slice, Figure 5.6. The scaled impulse response is also shifted to correspond to its time slot in the input signal, Figure 5.6B. Since LTI systems are, by definition, time invariant, time shifting the signal does not alter the impulse response. The output signal is the sum of all those scaled and shifted impulse responses. This approach to finding the output of a system given the input is called "convolution."

When implementing convolution, we reverse the input signal, because it is the input signal's smallest time value that produces the initial output. From a graphical perspective, the left-hand side of a time plot is actually the low-time side and it is this segment that enters the system first. So, from the point of view of the system, the left side of the input signal is encountered first and the leftmost segment produces the first impulse response, Figure 5.7. The input signal then proceeds through the system backward from the way it is normally plotted.

Since the segments are infinitesimal, there are an infinite number of impulse responses so the summation becomes integration. Standard calculus is well equipped to describe this operation, although implementation is another matter. Scaling and summing the time-shifted impulse responses generated by a reversed input signal leads to the "convolution integral" equation.

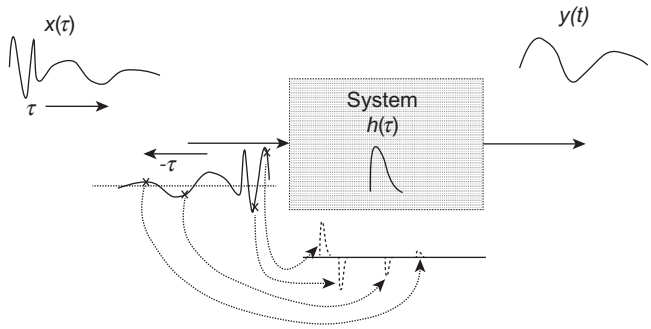


FIGURE 5.7 An input signal first enters its target system at its smallest (i.e., most negative) time value. From a graphical perspective, the first impulse response generated by the input signal comes from the left side of the signal as it is normally plotted. As it proceeds in time through the system backward, it generates a series of impulse responses that are scaled by the value of the input signal at any given time.

$$y(t) = \int_{-\infty}^{\infty} h(\tau)x(t-\tau)d\tau \tag{5.1}$$

where x is the input signal, $h(\tau)$ is the system's impulse response, and y is the output. Although the limits of integrations are $\pm\infty$, the actual limits are set by the length of the signals.

Convolution is the same as the cross-correlation equation (Equation 2.39) except for two alterations. First, the role of t and τ are switched so that t is now the time-shift variable and the second time variable, τ, is used for the input and impulse response functions. This modification is a matter of convention and is trivial. Second, there is a change in the sign between the time variables of x: $x(t+\tau)$ changes to $x(t-\tau)$. This is because the input signal, $x(\tau)$, must be reversed as described earlier, and the $-\tau$ accounts for this reversal. A third difference not reflected in the equations themselves is the way the two equations are used: cross-correlation is used to compare two signals, whereas convolution is used to find the output of an LTI system to any input.

As with cross-correlation, it is just as valid to shift the impulse response instead of the input signal. This leads to an equivalent representation of the convolution integral equation:

$$y(t) = \int_{-\infty}^{\infty} x(\tau)h(t-\tau)d\tau \tag{5.2}$$

The solution of Equation 5.1 or 5.2 requires multiple integrations, one integration for every value of t. Since t is continuous, that works out to be an infinite number of integrations. Fortunately, in real-world situations we apply convolution only in the discrete domain. In the discrete domain, t takes on discrete values at sample intervals T_s and integration becomes summation, so solving the convolution equation becomes a matter of performing a large number of summations. This would still be pretty tedious except that, as usual, MATLAB does all the work.

An intuitive feel for convolution is provided in Figures 5.8–5.10 as more and more impulse responses are added to the convolution process. Figure 5.8 shows the impulse response of a system in the left plot and the input to that system in the right plot. Note the different time scales: the impulse response is much shorter than the input signal, as is generally the case.

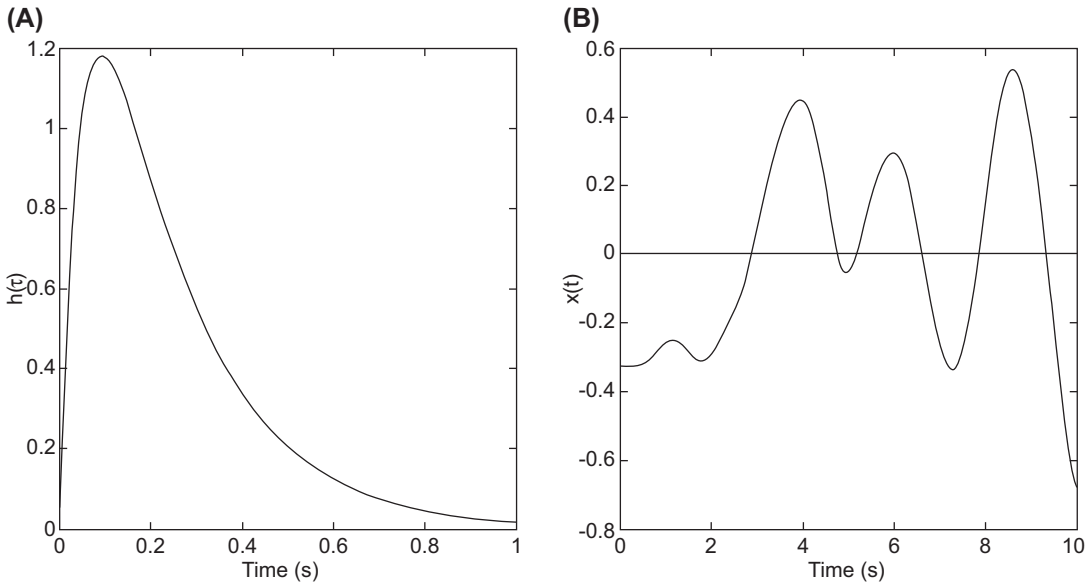


FIGURE 5.8 (A) The impulse response of a hypothetical system used to illustrate convolution. (B) The input signal to a system having the impulse response shown in (A). The impulse response has a much shorter duration than the input signal: 1 s versus 10 s. The impulse response is usually much shorter than the input signal.

In Figure 5.9A, the impulse responses to four signal segments at 2, 4, 6, and 8 s are shown. Each segment produces an impulse response that is shifted to the same time slot as the impulse, and the impulse response is scaled by the amplitude of the input signal at that time. Note that the input signal has been reversed as explained earlier. Some responses are larger,

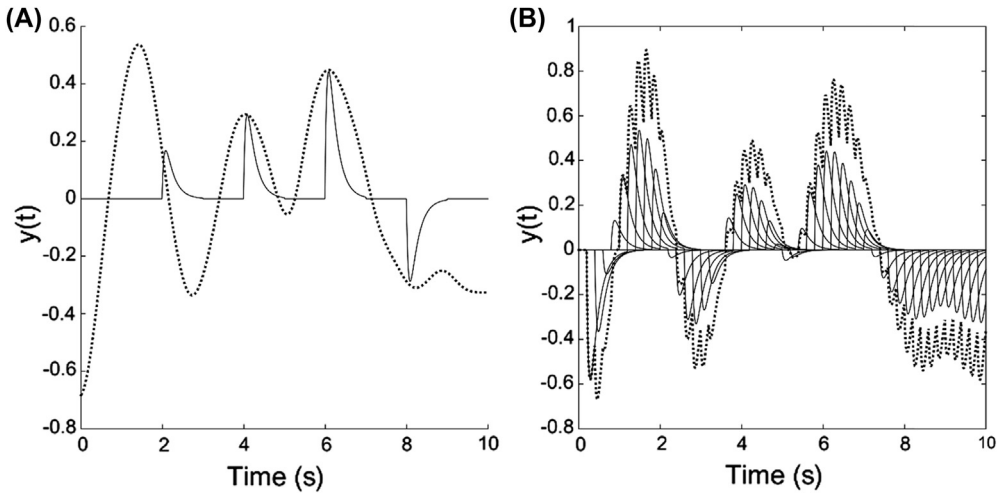


FIGURE 5.9 (A) The output response produced by four time slices of the input signal shown in Figure 5.6B. Impulse responses from input segments at 2, 4, 6, and 8 s are shown (*solid curve*) along with the reversed input signal, $x(-\tau)$ in Equation 5.2 (*dotted curve*). (B) Fifty impulse responses (*solid curves*) for the same reversed signal shown in (A). The sum of these signals is also shown (*dotted curve*).

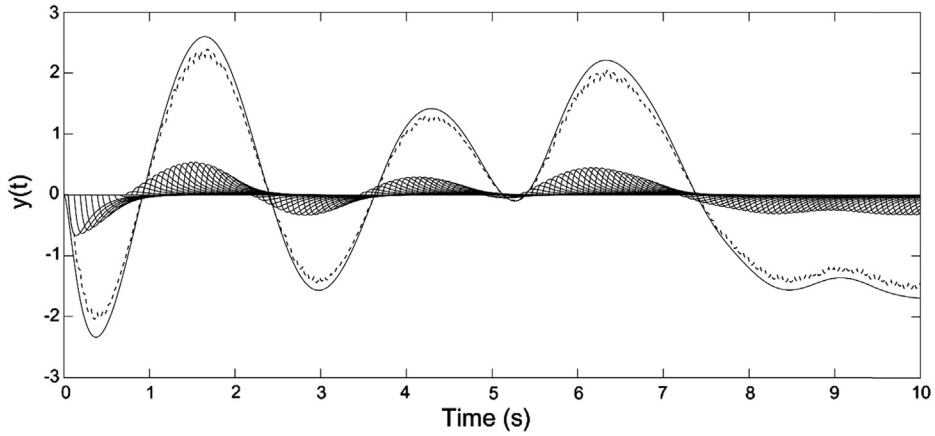


FIGURE 5.10 The summation of 150 impulse responses (*dashed curve*) from input segments spaced evenly over the input signal now closely resembles the actual output curve (*solid line*). The actual output is obtained from implementation of the digital convolution equation (Equation 5.3).

and one is scaled negatively, but they all have the basic shape as the impulse response shown in Figure 5.8A. A larger number (50) of these scaled impulse responses is shown in Figure 5.9B.

The 50 impulse responses in Figure 5.9 begin to suggest the shape of the output signal. A better picture is given in Figure 5.10, which shows 150 impulse responses, the summation of those responses (dashed line), and for comparison the actual output response (solid line) obtained by convolution. The summation of 150 impulse responses looks very similar to the actual output. (The actual output signal is scaled down to aid comparison.) As mentioned earlier, convolution of a continuous signal requires an infinite number of segments, but the digital version is limited to one for each data sample as discussed later. (The input signal used in the figures has 1000 samples.)

If the impulse response extends over a long period of time at high values, then a large contribution comes from past segments of the input signal. Such a system is said to have more memory, as the output signal is based on more of the input signal's past. If the impulse response is short, then very little comes from the past and more of the output is shaped by the instantaneous input at a given time. In the extreme case, if the impulse response is itself an impulse, then nothing comes from past inputs—all of the output comes from the current instantaneous input, and the output looks just like the input. The only difference would be the scale of the output, which would alter the height of the output impulse. Such a system is memoryless and would be called a "gain element."

For discrete signals, the integration becomes a summation and the discrete version of the convolution integral becomes the "convolution sum":

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] = \sum_{k=-\infty}^{\infty} h[k]x[n-k] \tag{5.3}$$

where n represents the time-shift variable and may extend over only the shorter of the two functions or, with zero padding, over all possible signal combinations. Again, it does not matter whether $h[k]$ or $x[k]$ is shifted; the net effect is the same. Since both $h[k]$ and $x[k]$ must be finite (they are stored in finite memory), the summation is also finite. The continuous and discrete convolution operations may also be abbreviated by using an "*" between the two signals:

$$y(t) = \int_{-\infty}^{\infty} h(\tau)x(t-\tau)d\tau \equiv h(t) * x(t) \tag{5.4}$$

or:

$$y[n] = \sum_{n=0}^{N-1} x[k]h[n-k] \equiv x[n] * h[n] \tag{5.5}$$

Unfortunately the * symbol is broadly used to represent multiplication in many computer languages, including MATLAB, so its use to represent the convolution operation can be a source of confusion and we avoid it here.

Superposition is a fundamental assumption of convolution. The impulse response describes how a system responds to a small (or infinitely small) signal segment. Each of these small (or infinitesimal) segments of an input signal generates its own little impulse response scaled by the amplitude of the segment and shifted in time to correspond with the segment's time slot. The output is obtained by summing (or integrating) the multitude of impulse responses, but this is valid only if superposition holds. Since convolution invokes superposition, it can be applied only to LTI systems where superposition is valid.

In the real world, convolution is done only on a computer, but in the textbook world, manual convolution can be found, in both the continuous and digital domains, as demonstrations. Sometimes these exercises provide insight and, in that hope, two examples of manual convolution are presented here. There are also a few in the problems.

EXAMPLE 5.3

Find the result of convolving the two continuous signals shown in Figure 5.11A and B. Use direct application of Equation 5.2. In this and the next example, one of the signals is assumed to be an impulse response $h(\tau)$ as is often the case, but of course convolution can be applied to any two signals.

The equations for the two signals can be determined from Figure 5.11:

$$x(\tau) = \begin{cases} 1 & 0 < \tau < 1.5 \\ 0 & \text{otherwise} \end{cases}; \quad h(\tau) = e^{-t} \quad 0 \le t < 1.5$$

Solution: First, one of the signals has to be flipped, then a sliding integration can be performed. In this example, we flip the impulse response as shown in Figure 7.6 C. So the basic signals in Equation 5.2 are the equation for x above and:

$$h(t-\tau) = \begin{cases} e^{-(t-\tau)} & 0<\tau<1.5 \\ 0 & \text{otherwise} \end{cases}$$

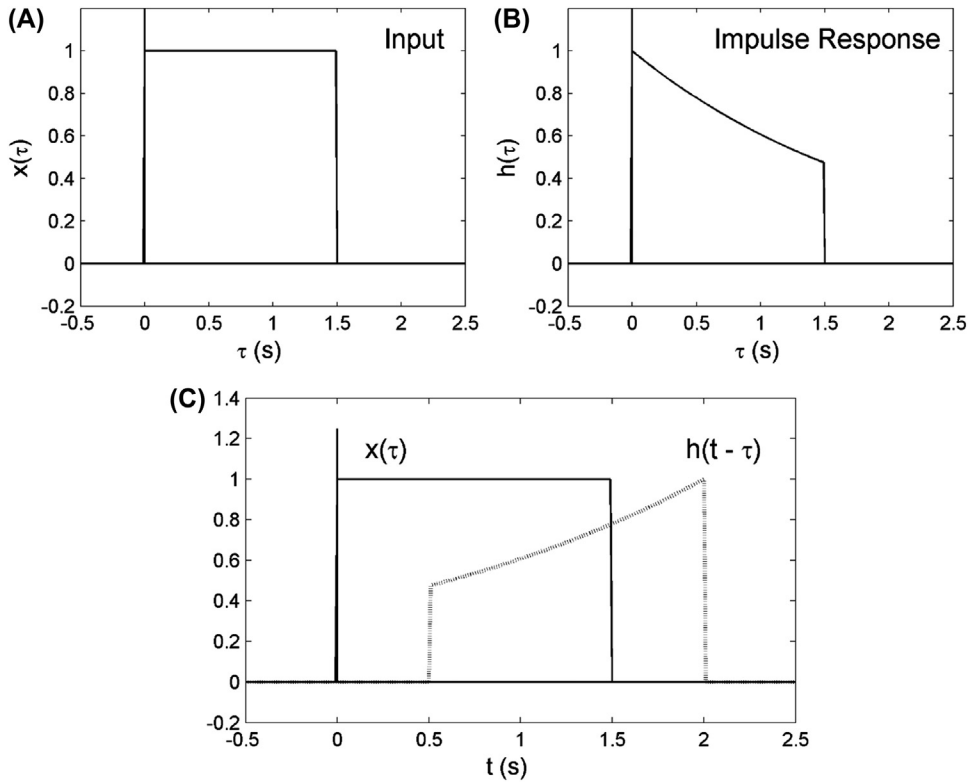


FIGURE 5.11 (A) and (B) Two signals to be convolved manually in Example 5.2. (C) To convolve the two signals, one signal, $h(\tau)$ is reversed. This reverse signal then slides across the other signal and the area of overlap is determined.

To solve the integral equation we use the classic calculus trick of dividing the problem into segments where the relationship between the two signals is consistent. For these two signals, there are three time periods during which the mathematical statement for $x(t)\ h(t-\tau)$ remains consistent, Figure 5.12.

In the first region, for shifts where $t < 0$, there is no overlap between the two signals, Figure 5.12A. So their product is zero and the integral is also 0:

$$y(t) = 0 \quad \text{for} \quad t < 0$$

When the time shift $t > 0$ s, the functions overlap to an extent determined by the impulse response, $h(t-\tau)$, as shown in Figure 5.12B.

$$y(t) = \int_0^{\infty} x(\tau)h(t-\tau)d\tau = \int_0^{t} 1e^{-(t-\tau)}d\tau = e^{-t}\left(e^{\tau}|_0^t\right) = e^{-t}e^{t} - e^{-t}e^{0} = 1 - e^{-t} \quad 0 \le t < 1.5$$

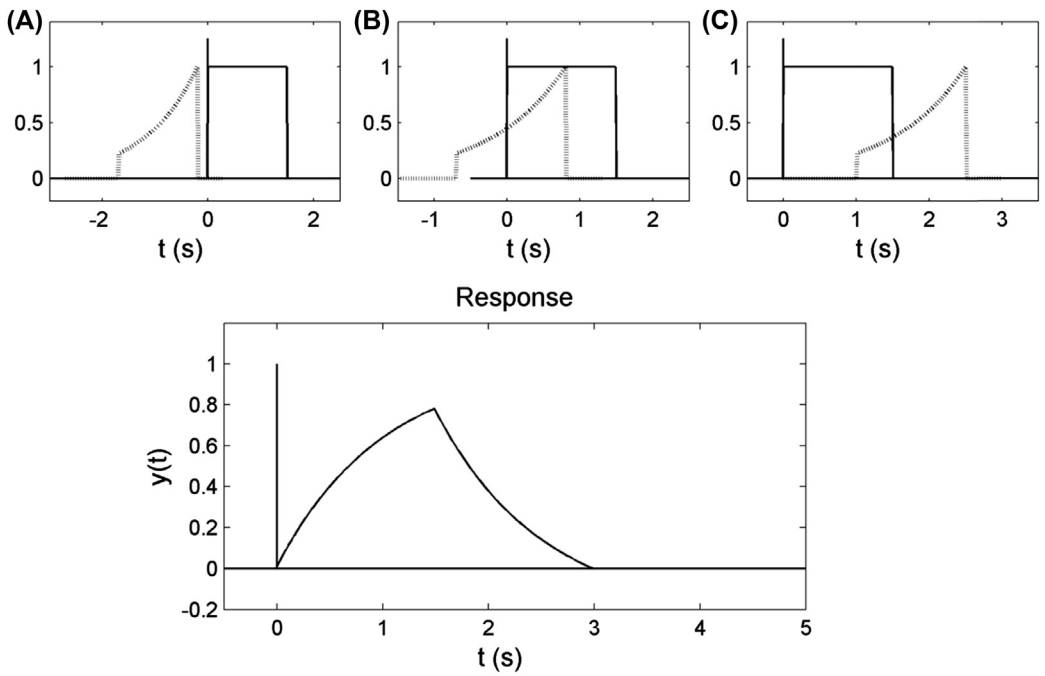


FIGURE 5.12 (A) For this relative position, where the shift time is $t < 0$, two signals have no overlap. (B) For the time shift where $0 \le t < 1.5$ s, the overlap depends on $h(t - \tau)$. (C) When $t > 1.5$ s, the overlap depends on the pulse function. (D) The result of the manual convolution of the signals shown in Figure 5.11.

This relationship continues until $t > 1.5$ s when the overlap between the two signals is determined by the pulse. For this region the integral is the same, except that the upper limit of integration is 1.5 s:

$$y(t) = \int_0^{1.5} 1e^{-(t-\tau)}d\tau = e^{-t}\left(e^{\tau}|_0^{1.5}\right) = e^{-t}e^{1.5} - e^{-t} = \left(e^{1.5} - 1\right)e - t \quad t \ge 1.5$$

Combining these three solutions gives:

$$y(t) = \begin{cases} 0 & t < 0 \\ 1 - e^{-t} & 0 \le t < 1.5 \\ \left(e^{1.5} - 1\right)e^{-t} & t \ge 1.5 \end{cases}$$

A time plot of the signal resulting from this convolution is shown in Figure 5.12D. An example of manual convolution with a step function that is 0 for $t < 0$ and a constant for all $t > 0$ is given in the problems.

Manual convolution of digital signals is comparatively easy; it is only a matter of keeping track of the shifts. The next example applies convolution to a digital signal.

EXAMPLE 5.4

Find the result of convolving the two discrete signals shown in Figure 5.13A and B. Use the direct application of the convolution sum in Equation 5.3. It is worth paying close attention to this example: if you understand what is going on in this example, you understand the basics of discrete convolution.

Solution: This example is similar to the digital cross-correlation shown in Example 2.15 except that one of the signals is flipped. The signal values for $x[k]$ and $h[k]$ are given in Figure 5.13. After reversing $h[k]$, we slide it across $x[k]$ one point at a time. At each position, we take the product of all the overlapping points and add them up. To handle the end points we pad $x[n]$ with two zeros on each end.

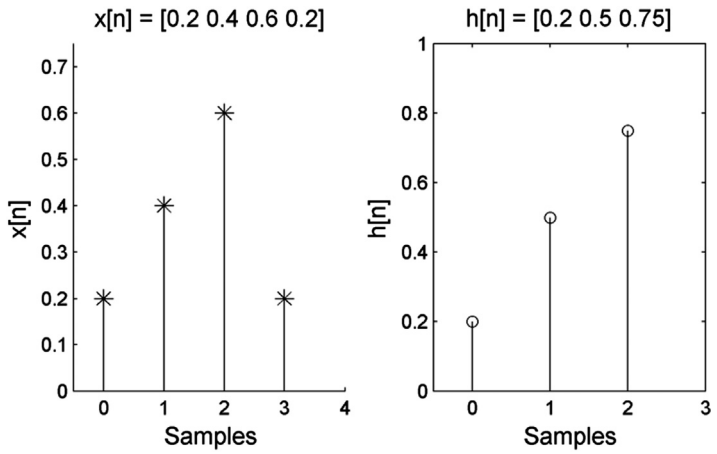


FIGURE 5.13 Two digital signals to be convolved manually in Example 5.4.

The operation is best illustrated graphically, Figure 5.14.

We begin with the flipped impulse response, ($h[-k]$, the "o" points) on the left side of the signal ($x[k]$, the "*" points): The first two "o" points overlap padded zeros; only the third point contributes to the convolution sum. So the first output point is

$x[k]$	0.0	0.0	0.2	0.4	0.6	0.2	0.0	0.0
$h[-k]$	0.75	.05	0.2					

$$y[1] = 0.2(0.2) = 0.04$$

After sliding the "o" points one position to the right, we now have two points overlapping nonzero values of the "*" points, giving the convolutions sum:

$x[k]$	0.0	0.0	0.2	0.4	0.6	0.2	0.0	0.0
$h[-k]$		.75	0.5	0.2				

$$y[2] = 0.5(0.2) + 0.2(0.4) = 0.18$$

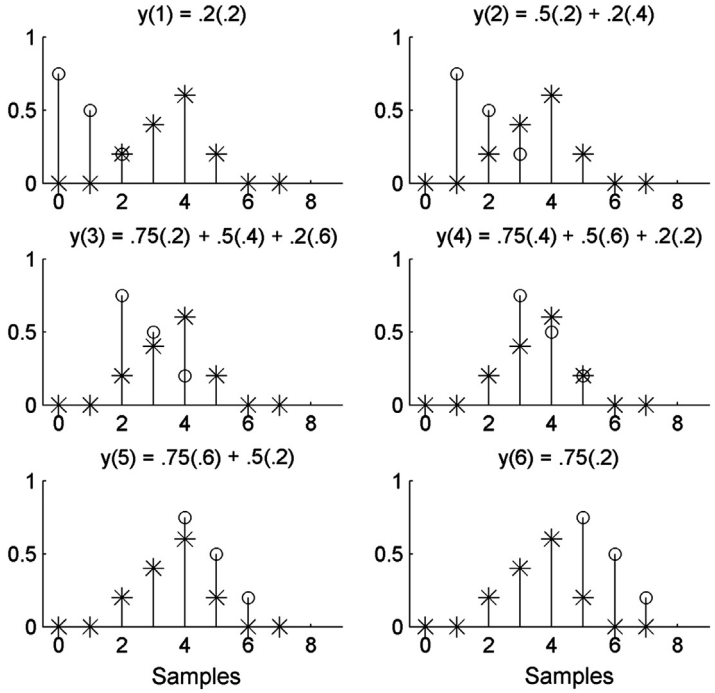


FIGURE 5.14 A step-by-step example of the manual convolution of the two digital signals shown in Figure 5.13. Note how $h[-k]$, the "o" points, slides across $x[k]$, the "*" points, producing one output point, $y[n]$, at each position.

For the next two positions, all three of the "o" points overlap nonzero "*" points, giving sums of:

$x[k]$	0.0	0.0	0.2	0.4	0.6	0.2	0.0	0.0
$h[-k]$			0.75	0.5	0.2			

$y[3] = 0.75(0.2)+0.5(0.4)+0.2(0.6) = 0.47$

$x[k]$	0.0	0.0	0.2	0.4	0.6	0.2	0.0	0.0
$h[-k]$				0.75	0.5	0.2		

$y[4] = 0.75(0.4) + 0.5(0.6) + 0.2(0.2) = 0.64$

For the last two positions of "o" points, first two points, then only one point, overlaps nonzero "*" points:

$x[k]$	0.0	0.0	0.2	0.4	0.6	0.2	0.0	0.0
$h[-k]$					0.75	0.5	0.2	

$y[5] = 0.75(0.6) + 0.5(0.2) = 0.55$

$x[k]$	0.0	0.0	0.2	0.4	0.6	0.2	0.0	0.0
$h[-k]$						0.75	0.5	0.2

$y[5] = 0.75(0.6) + 0.5(0.2) = 0.55$

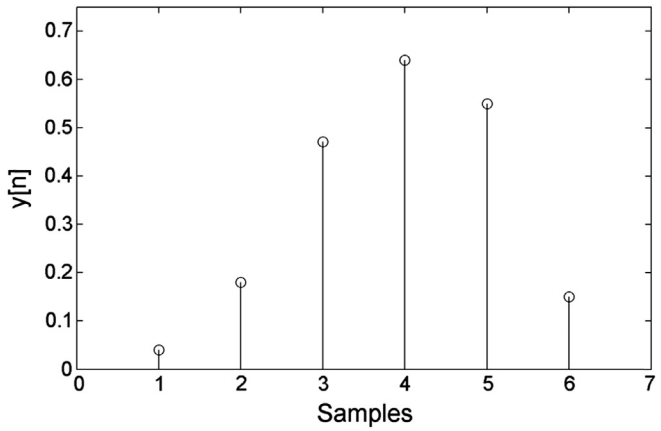


FIGURE 5.15 The result of the manual convolution of the two signals shown in Figure 5.11.

Results: The output signal $y[n]$ is shown in Figure 5.15. Although this may seem like a simple example since the number of points in both signals is small, it fully reflects the operations performed in all discrete convolutions. Being able to visualize this sliding summation (as $h[-k]$ slides to the right) is also helpful in understanding the operation of digital filters described in Chapter 8.

5.4.1 Finding the Impulse Response

Although convolution can be applied to any two signals, it is most commonly used to evaluate the output of a system to any input given the system's impulse response. Convolution is often used in signal processing to modify a signal in some controlled way. A common example is signal filtering where a system, the filter, is applied to a signal to remove unwanted frequencies. In such cases, the impulse response can be calculated based on the filter's desired frequency characteristics. We learn how to construct such impulse responses in Chapter 8.

Alternatively, if the system is physically available, as is often the case in the real world, the impulse response can be determined empirically by monitoring the response of the system to an impulse. Of course, a mathematically correct impulse function that is infinitely short but still has an area of 1.0 is impossible to produce in the real world, but an appropriately short pulse will serve just as well. Just what constitutes an appropriately short pulse depends on the dynamics of the system, and an example of how to determine if a pulse is a suitable surrogate for an impulse function was given in Example 5.1.

5.4.2 MATLAB Implementation

The convolution integral can, in principle, be evaluated analytically using standard calculus operations, but it quickly becomes tedious or impossible for complicated inputs. The discrete form of the convolution integral (Equation 5.3) is easy to implement on a computer. In fact, the major application of convolution is in digital signal processing, where it is frequently used to apply filtering to signals.

It is not difficult to write a program to implement Equation 5.3, but, of course, MATLAB has a routine to compute convolution, the `conv` routine:

```
y = conv(x,h,'option') ;        % The convolution sum (Equation 5.2)
```

where `x` and `h` are vectors containing the waveforms to be convolved, and `y` is the output signal. The options are given in Table 5.1.

If `'option'` is not given (or given as `'full'`), the length of the output signal is longer than the input signal (as in Example 5.4). When having a longer output signal is a problem, the '`same`' option computes all possible combinations, but outputs only the central portion of the result the same size as signal `x`.The option '`valid`' limits the summations to time shifts where `x` and `h` completely overlap returning a signal that is shorter than x.

TABLE 5.1 Options for the MATLAB `conv` routine

Option	Operation	Output Signal Length
None or `'full'`	Uses all possible relative positions Zero pads as necessary	`length(x) + length(h) −1`
`'same'`	Uses all possible relative positions with zero padding, but returns only the central sums	`length(x)`
`'valid'`	Sums over only relative positions possible without zero padding	`length(x) − length(h) −1`

An alternative routine, `filter`, can also be used to implement convolution. This routine always produces the same number of output samples as in `x`. When used for convolution, the calling structure is:

```
y = filter(h,1,x);      % Convolution sum (output same length as x)
```

where the variables are the same as described earlier and the second variable is always 1 as used here (more advanced applications of this routine are found in Chapter 8). In the following examples, the `conv` routine is used with the `same` option. You can try experimenting with using `filter` where appropriate in the problems. To get the proper output signal amplitude, both `conv` and `filter` require that the output signal be divided by the sampling frequency, f_s.

EXAMPLE 5.5

Find and plot the convolution sum for discrete versions of the two continuous signals used in Example 5.3.

Solution: Compared with the analytical solution in Example 5.3, this example is frightfully easy. We simply define the two signals in the digital domain and then apply the conv routine. With this routine, we need to normalize the output by f_s to get the proper scale. Since we use discrete versions of the continuous signals in Example 5.3, we do need to select a sample frequency. Figure 5.12D

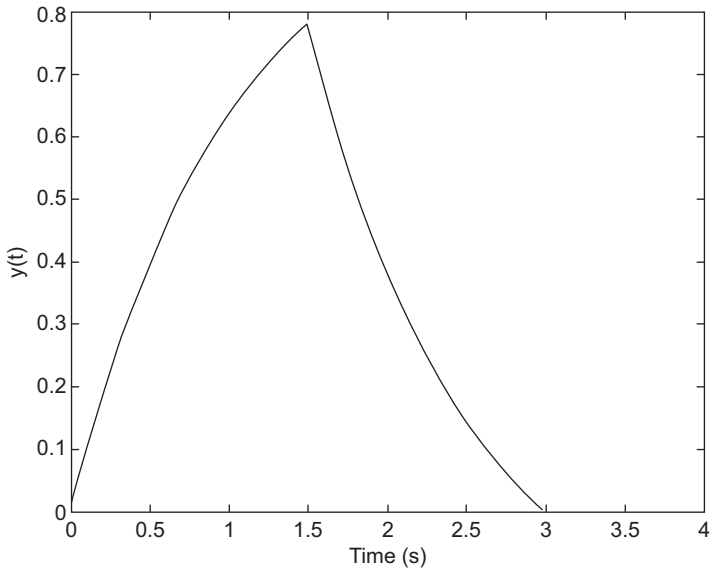


FIGURE 5.16 The convolution of the two signals in Figure 5.11 using MATLAB's `conv` routine. The resulting signal is the same as that found by manual calculation and plotted in Figure 5.12D.

shows that the output signal was 3 s long so if we choose an $f_s = 100$ Hz, our output signal will contain 300 samples, adequate for a decent plot. If we define `h` and `x` to be 1.5 s and use `conv` with the default option, the output will be approximately 3 s.

```
% Ex 5.5 Find the convolution of the two signals used in Example 5.3
%
fs = 100;                       % Sampling frequency
N = 150;                        % Number of samples in 1.5 sec
x = [ones(1,N)];                % Define pulse signal
t = (0:N-1)/fs;                 % Time vector for the exponential signal
h = exp(-t);                    % Define exponential signal
y = conv(x,h)/fs;               % Convolve and normalize
t = (0:length(y)-1)/fs;         % Time vector for plotting
     ........plot, label, and extend to 4 sec.......
```

Results: The result is shown in Figure 5.16 plotted out to 4 s. Using the default `conv` option outputs all possible points to give us the full response. The result is identical to the analytical solution found in Example 5.3 and shown in Figure 5.12D.

5.5 APPLIED CONVOLUTION—BASIC FILTERS

As mentioned, one of the most common uses of convolution is in signal processing to filter signals. The popularity of using convolution to implement signal filtering explains why MATLAB calls one of its convolution routines filter. The basic idea is to construct

impulse responses that when applied to a signal remove or reduce undesired frequencies. Again, we discover how to construct such impulse responses given the desired frequency characteristic in Chapter 9. The next example gives a taste of filtering using convolution.

EXAMPLE 5.6

Plot an electrocardiogram (ECG) signal before and after application of a filter (or system) consisting of a three-point running average. The ECG signal can be found as variable `ecg` in file `ecg_noise.mat`. $f_s = 125$ Hz. Also determine and plot the magnitude spectrum of the three-point running average system in decibels.

Solution: This example covers a lot of territory as it introduces the basic concepts behind a large class of digital filters: averaging across a series of sequential signal samples to construct the filtered signal. In this example of a three-point running average, we average three consecutive points, but in other filters we often apply a different weighting to each of the sequential points. We also get the spectral characteristics of this filter with the Fourier transform (and a lot of zero padding) as in Example 5.2.

A three-point running average (also known as a three-point moving average) constructs a filtered signal by taking three consecutive points from the unfiltered signal, averaging them to produce a new point in the filtered signal. This averaging process repeats after shifting one sample over the unfiltered data, Figure 5.17. The result is a smoother signal due to averaging.

We could easily write a program to perform a three-point running average. But looking back at the operations in Figure 5.14, a three-point running average is very similar to the sliding operation in discrete convolution. Convolution of the signal with an impulse response that consisted of three equal values of 1/3 each would effectively implement a three-point running average. In other words, a three-point moving average is equivalent to convolving a signal with an impulse response of $h = [1/3\ 1/3\ 1/3]$. When used in a filter, the individual samples of the impulse response are called the filter "weights," or filter "coefficients." Filters usually have short impulse responses.

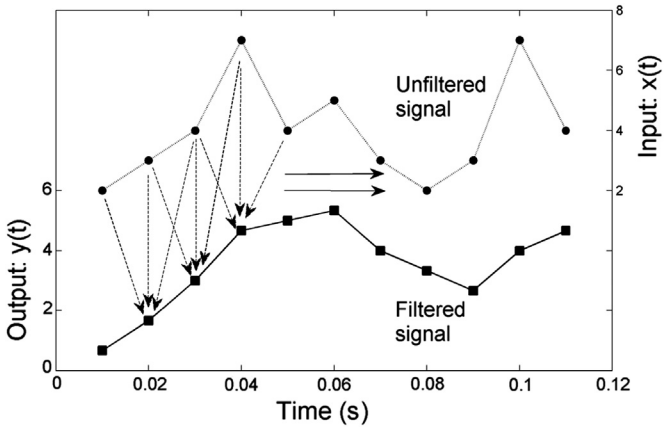


FIGURE 5.17 Filtering using a three-point running average. Each point on the filtered (lower) signal is constructed from the average of three consecutive points from the unfiltered signal. Endpoints are treated using zero padding. The three-point running average is identical to convolving the unfiltered signal with an impulse response having three equal values; i.e., $h = [1/3\ 1/3\ 1/3]$.

So after loading the data, we define the impulse response, carry out the convolution, and plot the results.

To find the equivalent spectrum of the three-point running average, note that in Section 5.3.2 we argued that a system's spectrum is the Fourier transform of the impulse response. Our three-point moving average is actually a system and its spectral characteristics are given by the Fourier transform of its impulse response, [1/3, 1/3, 1/3]. With only three points, the Fourier transform would be meaningless, so it is zero padding to the rescue. We will zero pad our short impulse response out to a large number of points to produce a smooth spectrum.

```
%  Example 5.6 Find the output of the ECG signal after filtering by a
%  filter that preforms a 3-point running average.
%
load ecg_noise;                    % ECG signal in variable eeg
N = length(ecg);                   % Number of data samples
fs = 125;                          % Sample frequency 125 Hz (given)
t = (1:N)/fs;                      % Construct time vector for plotting
h = [1 1 1]/3;                     % Define h(t) (3-point running avg)
out = conv(ecg,h,'same');          % Perform convolution
  ........ plot the filtered and unfiltered signal, new figure........
%
H = 20*log10(abs(fft(h,N)));       % Fourier transform of filter (heavily zero-padded)
N2 = round(N/2);                   % Plot only non-redundant points
f = (1:N)*fs/N;                    % Construct frequency vector for plotting
    ......plot the filter's spectrum.........
```

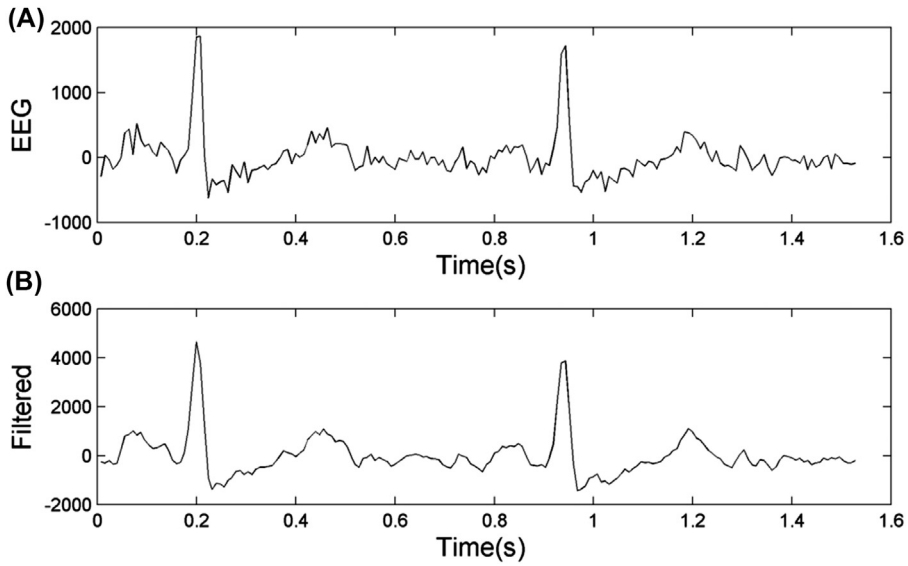


FIGURE 5.18 (A) A somewhat noisy unfiltered electrocardiogram (ECG) signal. (B) The ECG signal after filtering with the three-point running average filter.

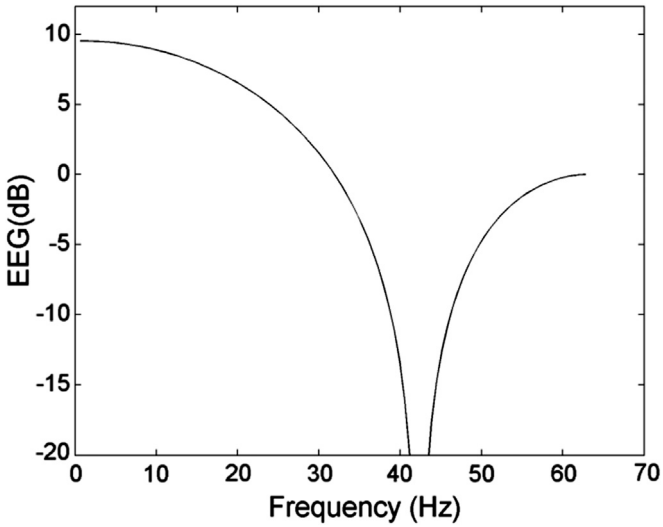


FIGURE 5.19 Magnitude spectrum of a three-point moving average filter.

Results: The ECG signal is shown before and after filtering in Figure 5.18. Note that the three-point running average smooths the signal slightly.

The filter's spectrum as obtained from the Fourier transform of the impulse response is shown in Figure 5.19. It has the form of a low-pass filter with a cutoff frequency of around 20 Hz. Reducing the higher frequencies reduces the spikey noise seen in the unfiltered ECG signal, Figure 5.18A.

You might wonder what would happen if we increased the number of sequential points that are averaged. What about a 10-point running average, or even a 20-point moving average? What would happen if not all the points in the impulse response had the same value? The not very informative answer is that then you would have a different filter. These questions are explored a bit in the problems and extensively in Chapter 8.

Just as signals and systems can be represented in either the time or frequency domain, the time-domain operation of convolution has a frequency-domain equivalent. To find out what the convolution operation would look like in the frequency domain, we use the same approach that we use to convert signals to the frequency domain: we apply the Fourier transform. We use the continuous representation of both convolution and the Fourier transform and will substitute ω for $2\pi f$ to make the equations cleaner. Starting with the convolution operation:

$$\int_{-\infty}^{\infty} h(\tau)x(t-\tau)d\tau = h(t) * x(t)$$

and taking the Fourier transform:

$$FT[h(t) * x(t)] = FT\left[\int_{-\infty}^{\infty} h(\tau)x(t-\tau)d\tau\right] = \int_{-\infty}^{\infty}\left[\int_{-\infty}^{\infty} h(\tau)x(t-\tau)d\tau\right] e^{-j\omega t}dt$$

Rearranging the order of integration:

$$= \int_{-\infty}^{\infty} h(\tau) \left[\int_{-\infty}^{\infty} x(t-\tau) e^{-j\omega t} dt \right] d\tau$$

The integral within the brackets is the Fourier transform of $x(t-\tau)$ a time-shifted version of $x(t)$. From the time-shift equation in Chapter 3 (Equation 3.41), that integral in brackets equals $X(\omega)e^{-j\omega\tau}$ (where $X(\omega)$ is the Fourier transform of $x(t)$). So the equation for convolution in the frequency domain becomes:

$$FT[h(t) * x(t)] = \int_{-\infty}^{\infty} h(\tau) X(\omega) e^{-j\omega\tau} d\tau = X(\omega) \int_{-\infty}^{\infty} h(\tau) e^{-j\omega\tau} d\tau = X(\omega) H(\omega) \tag{5.6}$$

Equation 5.6 shows that convolution of two signals in the time domain is the same as multiplying their frequency-domain representations. For example, if we had an impulse response of a system and wanted to find the output to a given signal, but for some reason we did not want to use convolution, we could multiply their frequency-domain representations instead:

1. Covert the impulse response and signal to the frequency domain by taking their Fourier transform: $H(\omega) = \text{FT}\,[h(t)]$ and $X(\omega) = \text{FT}\,[x(t)]$.
2. Multiply both: $Y(\omega) = H(\omega)\,X(\omega)$.
3. Take the inverse Fourier transform to get the time-domain output: $y(t) = \text{FT}^{-1}\,[Y(\omega)]$.

In the next example, we try out this strategy and compare it with straightforward convolution using MATLAB.[3]

[3]Insider tip: The roundabout way of performing convolution may seem needlessly complicated (one might even say convoluted), but it is the approach MATLAB uses in the `conv` routine! MATLAB uses this approach because the FFT algorithm and its inverse are so fast that it is actually faster to do convolution in the frequency domain going back and forth between the two domains than to execute all those time domain multiplications and additions called for by Equation 5.3.

EXAMPLE 5.7

Given the system impulse response in the equation below and shown in Figure 5.20B, find the output of the system to a periodic sawtooth wave having a frequency of 20 rad/s = 3.2 Hz, Figure 5.20A. Use both convolution and frequency-domain approaches. Since the sawtooth has a frequency of 16 Hz ($100/2\pi$), we use a sampling frequency that is 10 times greater: $f_s = 160$ Hz.

$$h(t) = 5.0\,e^{-5t} \sin(100t)$$

Solution: First construct the impulse response function given the equation above using a time vector of 1.0 s, Figure 5.20B. Finding the response using convolution requires only the application of MATLAB's `conv`. To find the output using the frequency-domain approach follow these steps: convert the impulse response and input signal to the frequency domain using the Fourier transform, multiply the two frequency-domain functions, and take the inverse Fourier transform.

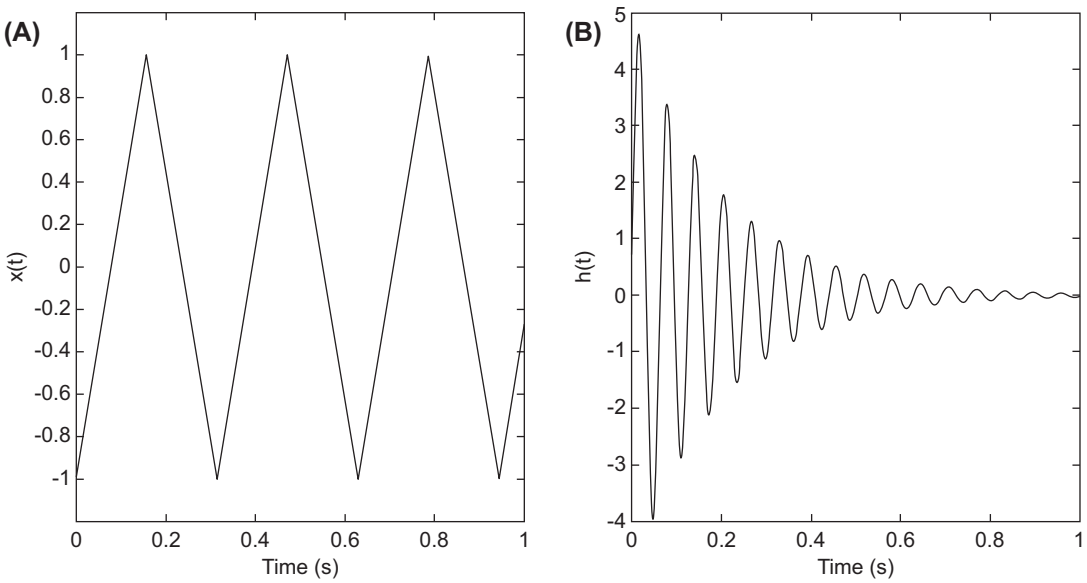


FIGURE 5.20 (A) Input signal to the system used in Example 5.7. (B) The impulse response of that system.

```
% Example 5.7  System output found two different ways.
%
fs = 160;                          % Sample frequency
T = 1;                             % Time in sec
t = (0:T)/fs;                      % Time vector
N = length(t);                     % Determine length of time vector
x = sawtooth(20*t,.5);             % Construct input signal
h = 5* exp(-5*t).*sin(100*t);      % Construct the impulse response
%
y = conv(x,h);                     % Calculate output using convolution
subplot(2,1,1)
plot(t,y(1:N));                    % Plot output from convolution
  .......label and title.......
% Now calculate in the frequency domain.
TF = fft(h);                       % Get TF from impulse response (Step 1)
X = fft(x);                        % Take Fourier transform of input (Step 1)
Y = X.*TF;                         % Take product in the frequency domain (Step 2)
y1 = ifft(Y);                      % Take the inverse FT to get y(t) (Step 3)
subplot(2,1,2)
plot(t,y1);                        % Plot output from frequency domain
    .......labels and title.......
```

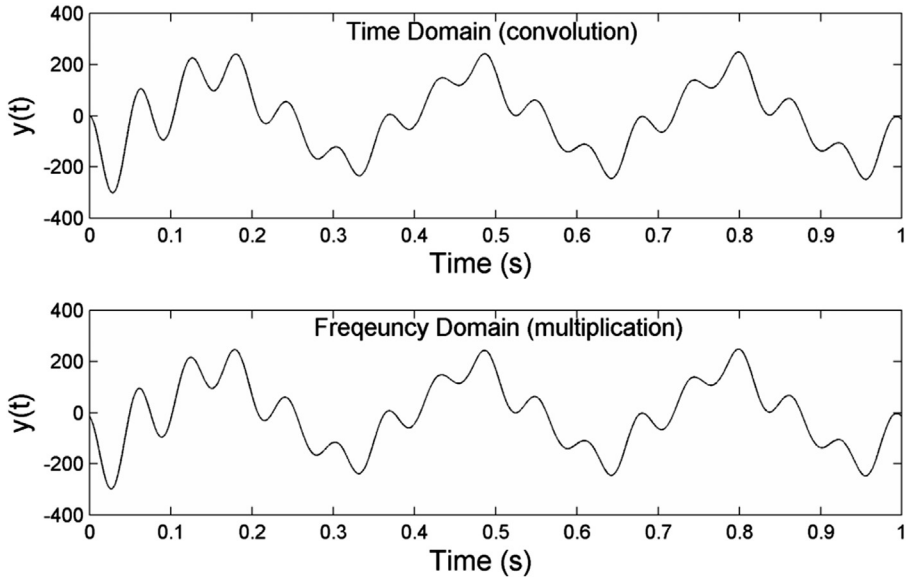


FIGURE 5.21 Output signals from a system defined by the impulse response in Figure 5.20B. These signals were computed using standard convolution (top) and multiplication of their frequency-domain representations (bottom).

Results: As shown in Figure 5.21, the output waveforms look like sinusoids riding on the input signal, and are the same using either time- or frequency-domain analysis.

5.6 CONVOLUTION IN THE FREQUENCY DOMAIN

Equation 5.6 states that convolution in the time domain is the same as multiplication in the frequency domain. The equations leading to Equation 5.6 can be rearranged to show that the reverse is also true: convolution in the frequency domain is the same as multiplication in the time domain.

$$X_1(\omega) * X_2(\omega) = x_1(t)\, x_2(t) \tag{5.7}$$

There are several important operations that involve time domain multiplication. In data sampling, a continuous signal is multiplied by a series of pulses spaced T_s apart (see Sections 4.2.1 and 1.2.3.2). Signal truncation is equivalent to multiplying the signal by a window function (see Section 4.2.3). Time domain multiplication is also found in amplitude modulation, where a signal is multiplied by a sinusoid or square wave. In the next section, we reexamine the sampling process specifically with respect to its influence on the signal's spectrum.

5.6.1 Data Sampling Revisited

In Chapter 4, we found that data sampling produces additional frequencies, in fact an infinite number of additional frequencies. The sampling process is like multiplying a continuous time function by a series of impulse functions. Figure 5.22A shows a continuous sinusoidal signal that is multiplied by a series of impulse functions spaced 50 ms apart. The result shown in Figure 5.22B is the same as a signal that is sampled at a frequency of $1/T_s = 1/0.05$ or $f_s = 20$ Hz. (A repeating series of pulses is referred to as a "pulse train.") So the sampling process can be viewed as a time domain multiplication of a continuous signal by a pulse train of impulses having a period equal to the sample interval, T_s.

Since multiplication in the time domain is equivalent to convolution in the frequency domain Equation 5.7), we should be able to determine the effect of sampling in a signal's spectrum using convolution. We just convolve the sampled signal's original spectrum with the spectrum of the pulse train. We get the former by applying the Fourier transform to the original signal. For the latter, the spectrum of a periodic series of impulse functions,

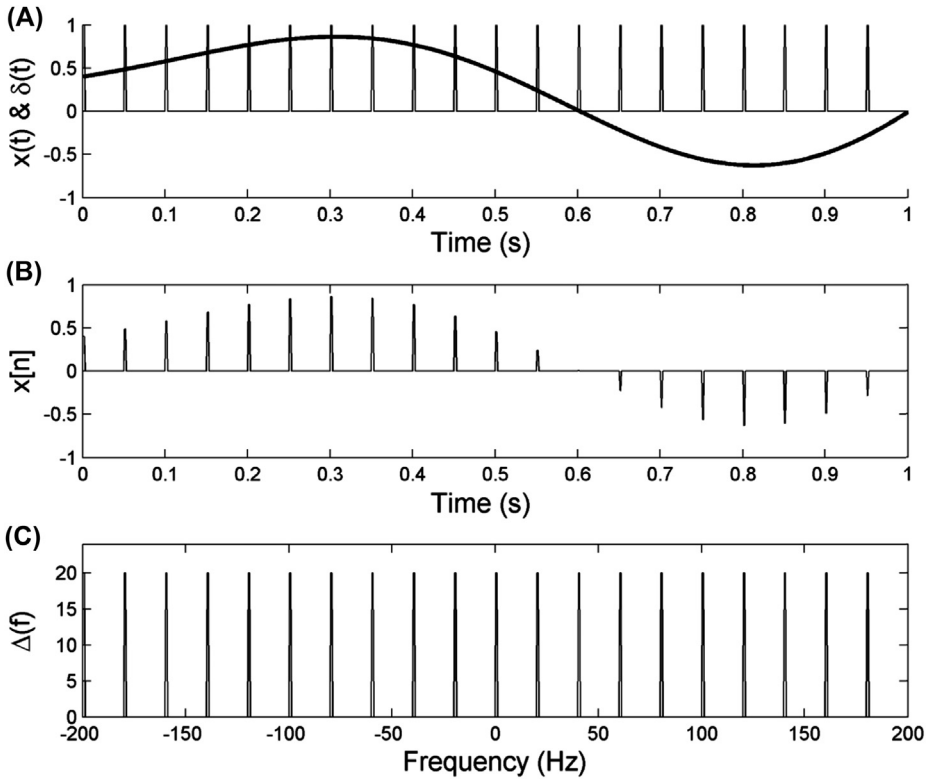


FIGURE 5.22 (A) The sampling process can be viewed as the multiplication of a continuous signal by a periodic series of impulse functions. In this figure the pulse train's period is 0.05 s, equivalent to the sample interval, T_s, or $1/f_s$. (B) The sampled signal, $x[n]$, where $f_s = 1/0.05 = 20$ Hz. (C) The frequency spectrum of a periodic impulse function is itself a series of impulse functions (Equation 5.10).

we use the Fourier series analysis, since the functions are periodic. In Example 3.3, the complex form of the Fourier series equation is used to find the Fourier coefficients of a periodic series of ordinary pulses having a pulse width W:

$$X_m = \left(\frac{V_p W}{T}\right) \frac{\sin(\pi m f_1 W)}{\pi m f_1 W} \quad m = 0, \pm 1, \pm 2, \pm 3, \ldots \tag{5.8}$$

For an impulse function $W \to 0$, but the area, $V_p W$, remains equal to 1.0. Also for small angles, $\sin(x) \approx x$, so as $W \to 0$ and $V_p W \to 1$, Equation 5.8 becomes:

$$\begin{aligned} X_m &= \lim_{\substack{WV_p \to 1 \\ W \to 0}} \left| \left(\frac{V_p W}{T}\right) \frac{\sin(\pi m f_1 W)}{\pi m f_1 W} \right| = \lim_{\substack{WV_p \to 1 \\ W \to 0}} \left| \left(\frac{V_p W}{T}\right) \frac{\pi m f_1 W}{\pi m f_1 W} \right| \\ &= \lim_{WV_p \to 1} \left| \frac{V_p W}{T} \right| = \frac{1}{T} \end{aligned} \tag{5.9}$$

Since the period of the impulse train is $T = T_s$, the spectrum of this impulse train interval is:

$$C_m = 1/T = 1/T_s = f_s \tag{5.10}$$

So the spectrum of the impulse function is itself a pulse train at frequencies of mf_1, where $f_1 = 1/T = f_s$ as shown in Figure 5.22C. Since the complex version of the Fourier series equation was used, the resultant spectrum has both positive and negative frequency values. Recall the impulse function is denoted by $\delta(t)$ so its spectrum is $\Delta(f)$ and is given as:

$$\Delta(f) = \frac{1}{T} \sum_{k=-\infty}^{\infty} \delta(f_s - k f_S) = f_s \sum_{k=-\infty}^{\infty} \delta(f_s - k f_S) \tag{5.11}$$

Now back to frequency-domain convolution. If the spectrum of the signal being sampled is $X(f)$ and the frequency spectrum of the impulse train is $\Delta(f)$, given by Equation 5.11, then the spectrum of the sampled signal is the convolution of the two. Applying the discrete convolution sum, Equation 5.3, and replacing time with frequency terms:

$$X_{samp}(f) = \Delta(f) * X(f) = \sum_{n=-\infty}^{\infty} \Delta(f) X(f - nf)$$

Substituting in Equation 5.11 for $\Delta(f)$, the spectrum of the sampled signal is:

$$X_{samp}(f) = \sum_{n=-\infty}^{\infty} \left(\frac{1}{T} \sum_{k=-\infty}^{\infty} \delta(f_s - k f_s) \right) X(f - nf) = \sum_{n=-\infty}^{\infty} \left(f_s \sum_{k=-\infty}^{\infty} \delta(f_s - k f_s) \right) X(f - nf) \tag{5.12}$$

After we rearrange the sums and simplify by combining similar frequencies, Equation 5.12 becomes:

$$XSamp = fs \sum_{k=-\infty}^{\infty} X(f - kfs) \tag{5.13}$$

This application of convolution in the frequency domain shows that the spectrum of a sampled signal is an infinite sum of shifted versions of the original spectrum. This is stated, but unproved, in Chapter 4.

5.7 SUMMARY

The impulse function, $\delta(t)$, is a very short pulse; in theory it is infinitely short, but also infinitely tall, so its area remains equal to 1.0. In the real world, whether a pulse can be considered an impulse depends on the response characteristics of the system it is used to evaluate. A pulse input is effectively an impulse if it produces a response that does not change in shape for incremental changes in pulse width. The response of such a short pulse is called the impulse response and it has a unique shape that does not depend on the amplitude of the pulse: it simply scales up or down proportionally with pulse amplitude. Since the shape of the impulse response is always the same for a particular system, it can be used as a descriptor of that system. Impulse responses can be determined empirically if the system is available by monitoring the system's response to a sufficiently short pulse. The most useful feature of the impulse and impulse response is that it can be used in a time-slicing approach to determine the output of a system to any input using convolution.

Convolution employs time slicing to determine the response of a system to any input signal. All you need is the system's impulse response, which can be considered the system's response to an infinitesimally short segment of the input. If the system is linear (LTI) and superposition holds, the impulse response from each input segment can be summed to produce the system's output. The convolution integral and convolution sum (Equations. 5.1, 5.2, and 5.3) are basically running correlations between the input signal and the impulse response. This integration can be cumbersome to compute analytically, but is easy to program on a computer. MATLAB provides two routines, `conv` and `filter`, to perform convolution.[4]

Convolution is commonly used in signal processing to implement digital filtering. In these applications, the impulse response of the desired filter is first determined analytically using techniques described in Chapter 8. The filtered signal is obtained by applying the filter's impulse response to the signal using convolution.

Convolution performs the same function in the time domain as multiplication in the frequency domain. The reverse is also true: convolution in the frequency domain is like

[4]The `conv` routine actually operates in the frequency domain: it converts both the impulse response and input signal to the frequency domain, multiplies the two, then converts back to the time domain. It is actually faster than the `filter` routine.

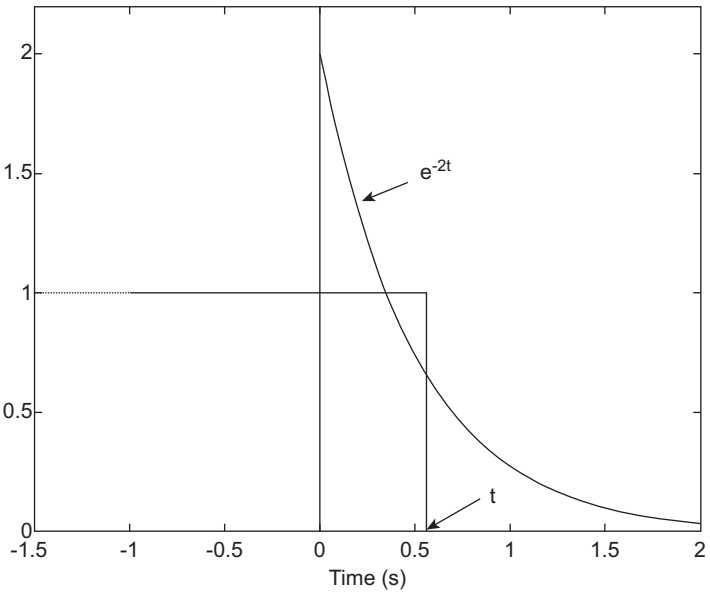
multiplication in the time domain. Frequency-domain convolution is useful in explaining time-domain operations such as sampling. Specifically, frequency-domain convolution explains the additional frequencies found in the spectra of time sampled signals. Recall that if the frequencies of a digitized signal exceed the Nyquist frequency, $f_s/2$, these added frequencies produce aliasing hopelessly corrupting the signal's spectrum.

PROBLEMS

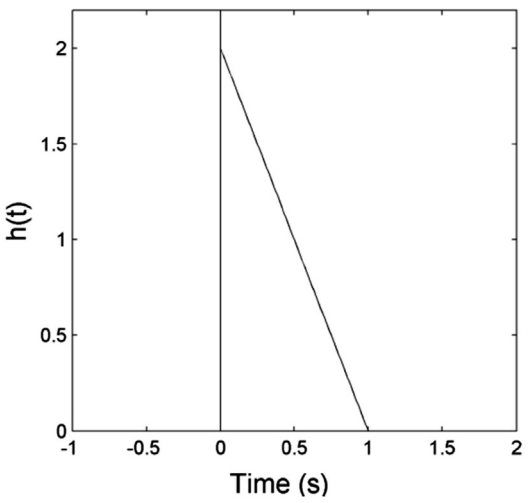
1. Modify the code in Example 5.1 to find the maximum length of a pulse that will serve as an impulse to the unknown system, `unknown_sys5_2`. Note this system has quite different response characteristics and the maximum pulse width will also be quite different. (Hint: You need to look at complete impulse response to determine the maximum pulse width. This system is much slower than that in Example 5.1, so you need to substantially increase the time length of the plots to capture the full impulse response.)
2. Using the pulse width obtained for an unknown_sys5_2 in Problem 1, show that the impulse responses to a wide range of pulse amplitudes have exactly the same shape. Use three different pulse inputs ranging in amplitude by a factor of 100. (Hint: Scale the responses to be the same maximum amplitude and show that they are identical.)
3. A pulse having a width of 40 ms is known to be sufficiently short to be an impulse input for the system `unknown_sys5_3`. Plot the impulse response in the time domain and be sure to use a long enough time period to accurately capture the complete impulse response.

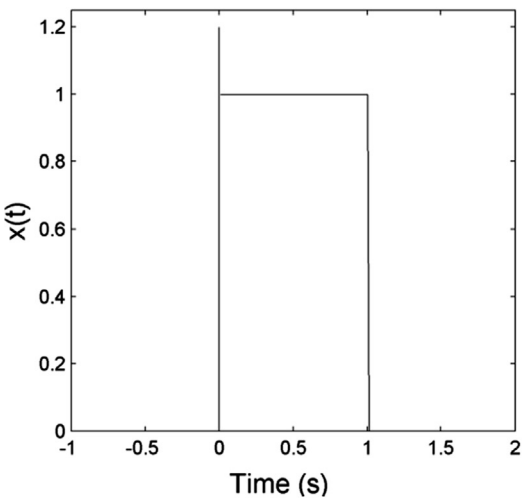
   Convert the system's impulse response and the 40-ms pulse to the frequency domain. Plot the magnitude spectra of the two superimposed to show that the 40-ms pulse has an nearly flat magnitude spectrum over the frequencies to which the system is capable of responding, i.e., the frequencies in the impulse response. Limit the frequency axis to a range that clearly shows the impulse response frequencies. If you plot the two spectra on the same graph you need to scale one or both to have similar ranges.
4. The `file unknown_impulse_response.mat` contains the impulse response of system in variable `i_r`. The impulse response was sampled at $T_s = 0.01$ s ($f_s = 100$ Hz.) Use this impulse response to find the magnitude spectrum of the unknown system. Then find the longest pulse that will still serve as an impulse to that system based on the pulse spectra. When generating the pulse, use the same data length as the impulse response. (Hint: The magnitude spectrum of the pulse should be relatively flat in the nonzero region of the system. You can plot the spectra of several pulses superimposed to speed up the search for maximum pulse width. When plotting the spectra, restrict the frequency axis to the region where the impulse response spectrum is nonzero. Finally, in reporting the maximum pulse width, note the sample interval and report the maximum pulse width in s.)
5. Use the basic convolution equation (Equation 5.1) to find the output of a system having an impulse response of $h(t) = 2\,e^{-10t}$ to an input that is a unit step function; i.e., $x(t) = 1$). $t > 0$. (Hint: If you reverse the step input, then for $t < 0$ there is no overlap

and the output is 0. For $t > 0$, the integration limit is determined only by t, the leading edge of the reversed step function, so only a single integration, with limits from 0 to t, is required. See the following figure.)

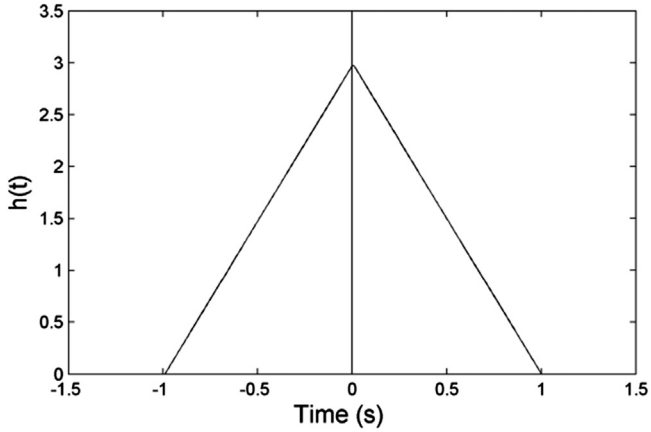


6. Use the basic convolution equation (Equation 5.1) to find the output of a system with an impulse response, $h(t) = 2(1 - t)$, to a 1-sec pulse having an amplitude of 1.

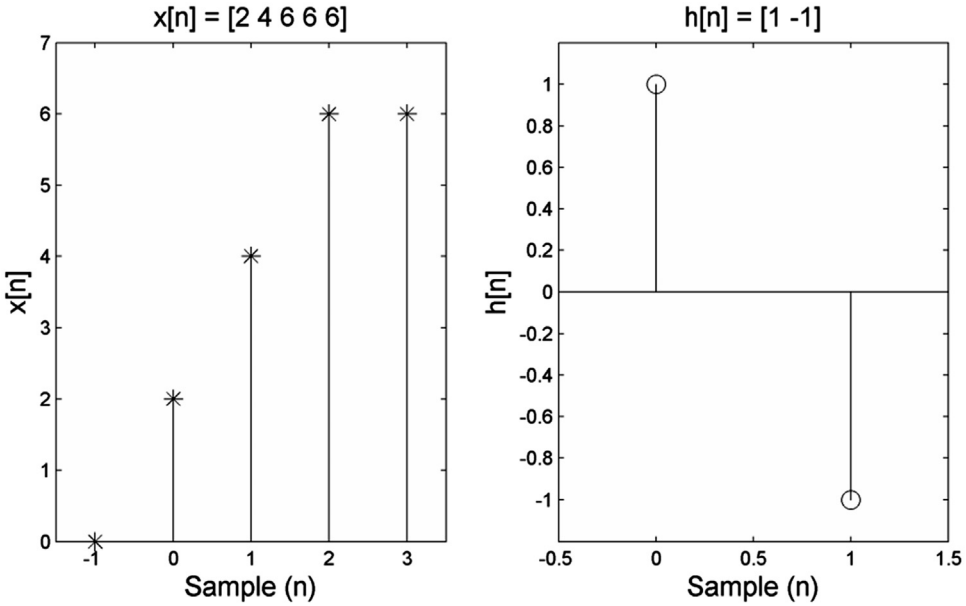




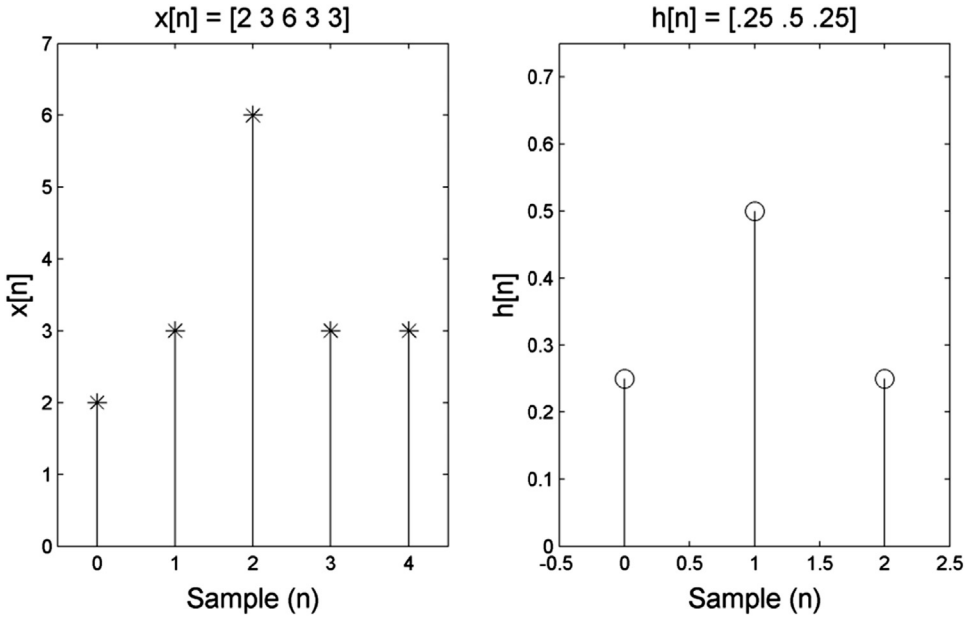
7. Use the basic convolution equation (Equation 5.1) to find the output of a system with an impulse response shown below to a step input with an amplitude of 5; i.e., $x(t) = 5$ for $t > 0$.



8. Find the convolution sum (Equation 5.3) for the discrete impulse response and discrete input signal shown in the following figure.



9. Find the convolution sum (Equation 5.3) for the discrete impulse response and discrete input signal shown in the following figure.



10. Systems known as filters are designed to alter the spectral characteristics of signals. In Example 5.6, we saw how a filter represented by an impulse response of three equal weights smoothed a noisy signal. What if we used more weights? What if we used weights that were not all the same? These questions are addressed (at least in part) by this and the next four problems.

Compare the frequency characteristic of three impulse responses that could be used as filters. The first impulse response consists of three equal weights used in Example 5.6. The second consists of five equal weights of 1/5. (Thus it computes a five-point running average.) The third filter consists of five different weights: `h3 = [0.0264 0.1405 0.3331 0.3331 0.1405 0.0264]`. To find the filter's spectrum, take the Fourier transform of these impulse responses. Be sure to sufficiently zero pad (and make N a power of 2 for a faster FFT while you are at it). As always, plot only the valid points and be sure the frequency axis is correctly scaled.

You will see that they are all low-pass filters and that, although the two equal-weight filters cut off more sharply, the unequal impulse response filter gives a much smoother frequency characteristic. We will learn how to design these filter impulse responses in Chapter 8.

11. There is more than one way to identify the magnitude spectrum of a system. The approach described in Section 5.3.2 and used in Problem 10 is to convert the impulse response to the frequency domain. Another way is to use sinusoidal inputs to the

system: multiple sinusoids that cover the frequency range of this system. Use convolution to find the system's output to these sinusoids, then plot the output amplitude divided by the input amplitude as a function of frequency. In the next three problems we use both approaches.

In this problem, we examine the four-coefficient "Daubechies filter":

$$h[n] = [0.683 \quad 1.183 \quad 0.3169 \quad -.0183]$$

A. Determine and plot magnitude frequency characteristics of this filter by taking the Fourier transform of the impulse response. Use an $f_s = 100$ Hz.[5] Be sure to pad sufficiently and plot frequencies between 0 and $f_s/2$.

B. Set up a loop to evaluate the output of this filter to 50 sinusoids varying in frequency from 1 to 50 Hz in 1.0-Hz intervals; i.e., from 1 to $f_s/2$ Hz. At each of the 50 frequencies, generate a sine wave (suggested N of 256), convolve it with $h[n]$ above, and take the root mean square (rms) value of the output. Plot that value at the frequency of the sine wave. (If you make the peak-to-peak amplitude of the sine wave input 1.414, it will have an rms value of 1.0 and you will not need to divide the output by the input.)

Use `subplot` to plot the two magnitude spectra. If done correctly, they should be nearly identical. Note that this is a low-pass filter with very gradual, smooth attenuation. This filter has some very special properties and is one member of a family of filters used in Wavelet analysis.

12. Repeat Problem 11 using one of the simplest of all filters, the "Haar" filter. This filter has an impulse response consisting of only two coefficients each with a value of 1/2:

$$h[n] = \left[\frac{1}{2} \quad \frac{1}{2}\right]$$

Although the Haar filter has a gradual attenuation and is not a very strong filter, it is used to demonstrate some basic principles of Wavelet analysis.

13. Load the respiratory signal `resp` in file `resp_noise.mat` ($f_s = 125$ Hz). Apply a 3-point moving average filter and a 12-point moving average filter. For variety, use the MATLAB filter routine:

(`y = filter(h,1,x);`), which produces a filtered signal that is the same length as the input signal. Display the results from the two filters in two plots using `subplot`. Each plot should compare the unfiltered signal with one of the filtered signals. Title the plots and offset the filtered signal (by 1.0) for better viewing. Note the improved noise reduction with the 12-point moving average filter.

14. So far we have examined only the magnitude spectrum of our filters. Sometimes it is important to know the phase characteristic of the filter. The phase spectrum of any system can be obtained directly from the Fourier transform of the impulse response

[5]It does not matter what f_s you use, you get the exact same spectral shape if you plot the entire range of valid frequencies, 0 to $f_s/2$. This is because f_s cancels out in the DFT equation. See Equation 3.30.

using the MATLAB `angle` routine. Usually you also have to apply MATLAB's `unwrap` routine to get the correct phase spectrum.

Show both the magnitude and phase spectra of two filters, one a four-point moving average and the other the four-coefficient "Daubechies" filter used in Problem 11. Use $f_s = 200$ Hz and adequate padding. Plot phase in degrees and plot only valid points. Use subplot to bring the four plots together.

Note that, although the four-point moving average has a sharper cutoff, the Daubechies filter has a smoother phase characteristic.

15. Load the MATLAB file `filter1.mat`, which contains the impulse response, `h`, of a mystery filter. Also load the signal in `cardiac_press.mat`, which holds one cycle of a cardiac pressure wave in variable `c_press` $f_s = 200$ Hz. Apply the impulse response, `h`, to the pressure signal using convolution. Plot the cardiac pressure signal before and after filtering on separate plots using subplot. What does this mystery filter do?

16. Repeat Problem 11 using the impulse response, `h`, in file `bp_filter.mat`. This is the impulse response of a band-pass filter. This is a narrowband filter, so to get an accurate spectrum using sinusoids you should make the frequency increments 0.5 Hz instead of 1.0 Hz. Then you need to increase the number of sine waves to 100 to get the full valid spectrum.

17. Repeat Problem 11 using the impulse response, `h`, in file `x_impulse.mat`. This is a resonant system with a sharp peak at low frequencies. To get an accurate spectrum using sine waves, decrease the frequency increment to 0.1 Hz, but use only 50 sine waves covering the range of 0−5 Hz. When taking the Fourier transform, you may have to increase the padding to get the correct magnitude spectrum.

18. This problem demonstrates that the spectrum of the sampling process, essentially a pulse train, is itself a pulse train. To simulate the sampling process, construct a 16384-point signal (2^{14}) of zeros then add a pulse of 1.0 at every 16th position (i.e., `pt(1:16:end) = 1)`. Take the Fourier transform and plot. If you do not scale the horizontal axis, you should see a peak at every 1024 points. Assuming the horizontal axis is in Hz, this corresponds to an $f_s = 1024$. Now simulate halving the sampling frequency by setting every 32nd point of the sampling signal to 1.0. You will now see peaks every 512 Hz reflecting the lower sampling frequency. Although you do not need to scale the axes, they should still be labeled. Combine the plots using `subplot` and title them for clarity.

19. This problem simulates the effect of sampling frequency on a spectrum. The spectrum can be found in file `unknown_spectrum.mat` as variable `Spec`. This spectrum represents the true spectrum of a signal before sampling. Plot this spectrum without scaling the frequency axis. Simulate the Fourier transform of a sampling process where $f_s = 2000$ Hz. Construct a 10,000-point array and make every 2000th point 1.0 (i.e., `pt(1:2000:end) = 1;`). Convolve this array with the spectrum `Spec` and plot the results. This is the spectrum you would get when sampling the signal at 2000 Hz. Now lower the sampling frequency to 1000 Hz by making every 1000th point of the sampling spectrum 1.0. Convolve and plot the results. Note the confused spectra that result from this lower sampling frequency, a condition described in Chapter 4 known as aliasing. Although you do not need to scale the axes, they should still be labeled. Combine the plots using subplot and title them for clarity.