Course: Digital Signal and Data Processing

Recursive Filters

Lecture 5

Associate professor Naila Allakhverdiyeva

The Recursive Method

Recursive filters are an efficient way of achieving a long impulse response, without having to perform a long convolution. They execute very rapidly, but have less performance and flexibility than other digital filters.

Recursive filters are also called **Infinite Impulse Response (IIR) filters**, since their impulse responses are composed of decaying exponentials. This distinguishes them from digital filters carried out by convolution, called ***Finite Impulse Response*** **(FIR)** filters.

The Recursive Method

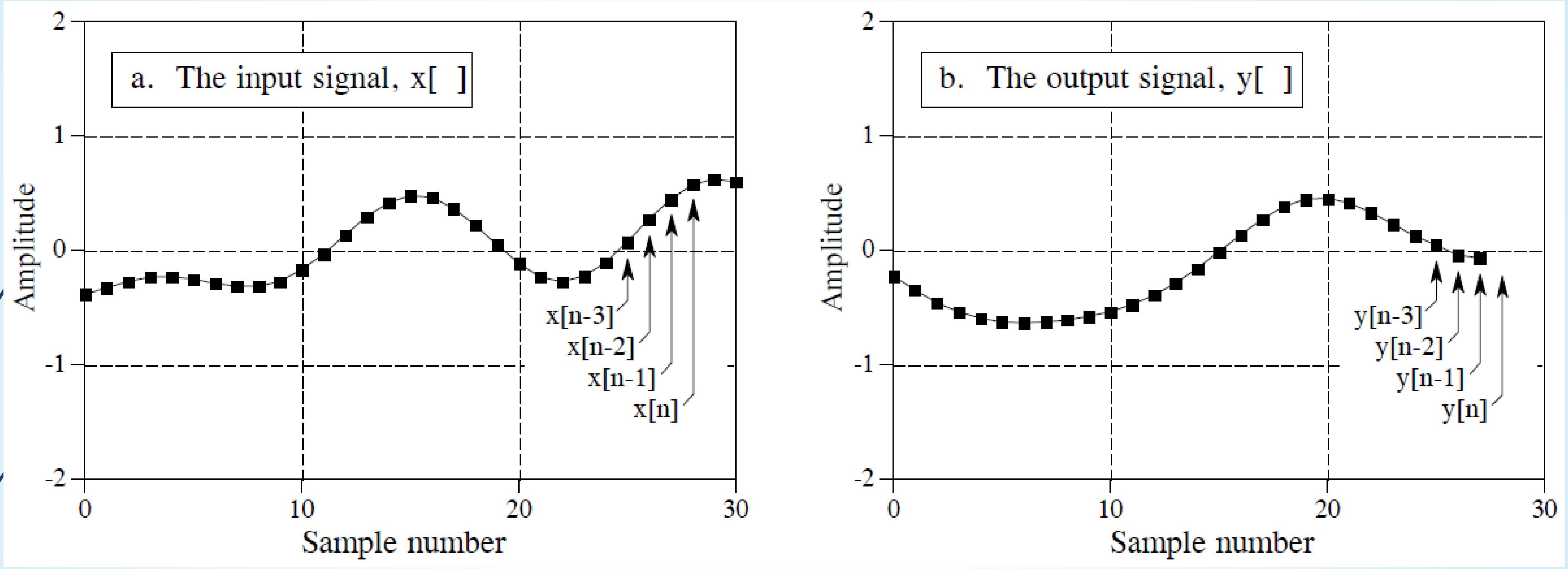
$$\begin{aligned} y[n] &= a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] + a_3 x[n-3] + \cdots \\ &\quad + b_1 y[n-1] + b_2 y[n-2] + b_3 y[n-3] + \cdots \end{aligned}$$

Each point in the output signal is found by multiplying the values from the input signal by the "$\boldsymbol{a}$" coefficients, multiplying the previously calculated values from the output signal by the "$\boldsymbol{b}$" coefficients, and adding the products together. Notice that there isn't a value for $\boldsymbol{b_0}$, because this corresponds to the sample being calculated.

This Equation is called the **recursion equation**, and filters that use it are called **recursive filters**. The "$\boldsymbol{a}$" and "$\boldsymbol{b}$" values that define the filter are called the **recursion coefficients**.

In actual practice, no more than about a dozen recursion coefficients can be used or the filter becomes unstable (i.e., the output continually increases or oscillates).

The Recursive Method



The output sample being calculated, $y[n]$, is determined by the values from the input signal, $x[n], x[n-1], x[n-2], \dots$, as well as the *previously* calculated values in the output signal, $y[n-1], y[n-2], y[n-3], \dots$. These figures are shown for $n = 28$.

The Recursive Method

Recursive filters are useful because they ***bypass*** a longer convolution.

For instance, consider what happens when a delta function is passed through a recursive filter. The output is the filter's ***impulse response***, and will typically be a *sinusoidal oscillation that exponentially decays*. Since this impulse response in infinitely long, recursive filters are often called **infinite impulse response (IIR)** filters.

Recursive filters ***convolve*** the input signal with a very long filter kernel, although only **a few coefficients are involved**.

The Recursive Method

The relationship between the recursion coefficients and the filter's response is given by a mathematical technique called the **z-transform**.

$$H[z] = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + \cdots}{1 - b_1 z^{-1} - b_2 z^{-2} - b_3 z^{-3} - \cdots}$$

For example, the z-transform can be used for such tasks as: converting between the recursion coefficients and the frequency response, combining cascaded and parallel stages into a single filter, designing recursive systems that imitate analog filters, etc.

Unfortunately, the z-transform is very mathematical, and complicated.

The Recursive Method

There are **three ways** to find the recursion coefficients **without** having to understand the z-transform.

- via equations for several types of simple recursive filters.
- though designing the more sophisticated *Chebyshev* low-pass and high-pass filters.
- Using an iterative method for designing recursive filters with an ***arbitrary*** frequency response.

Single Pole Recursive Filters

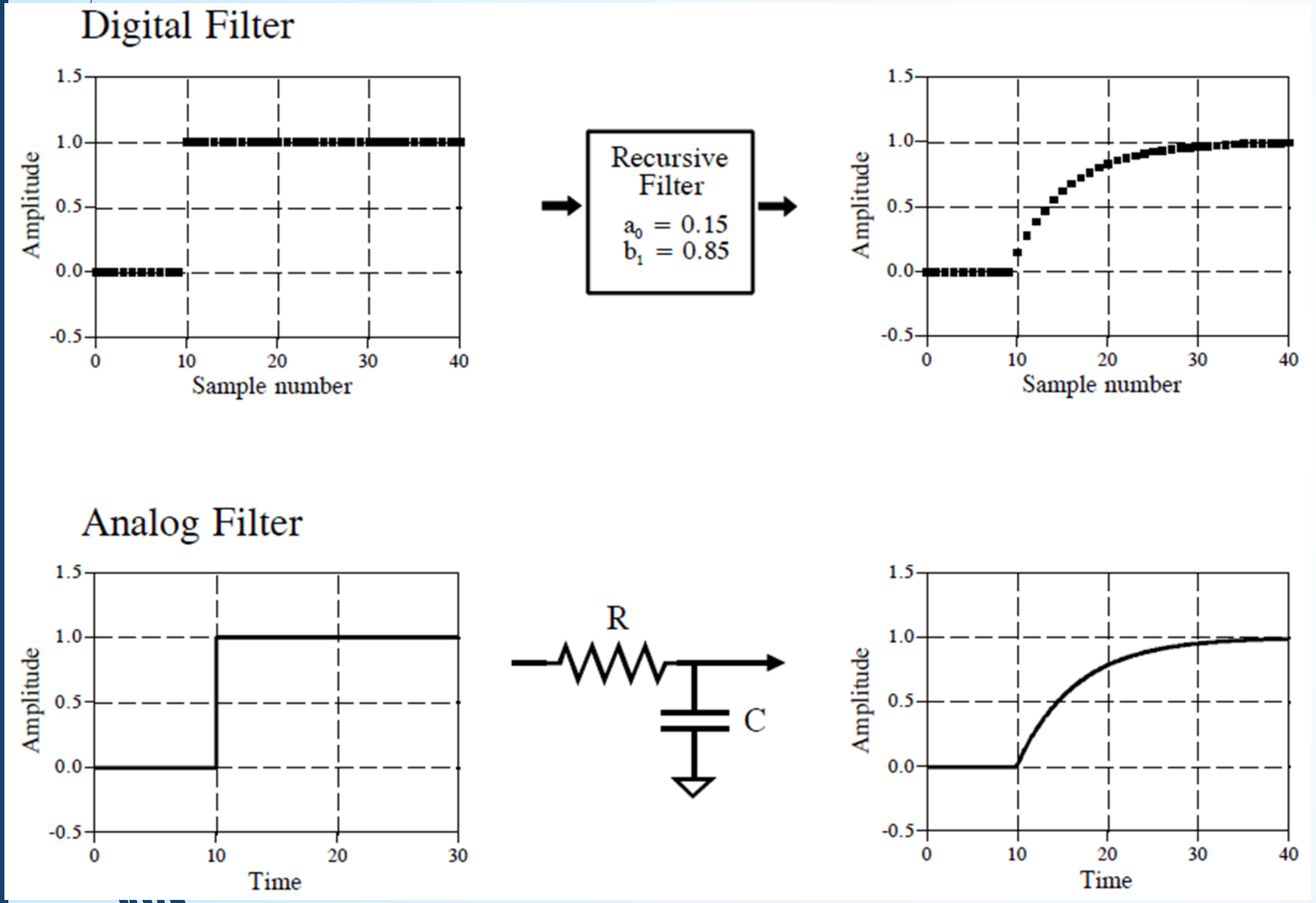
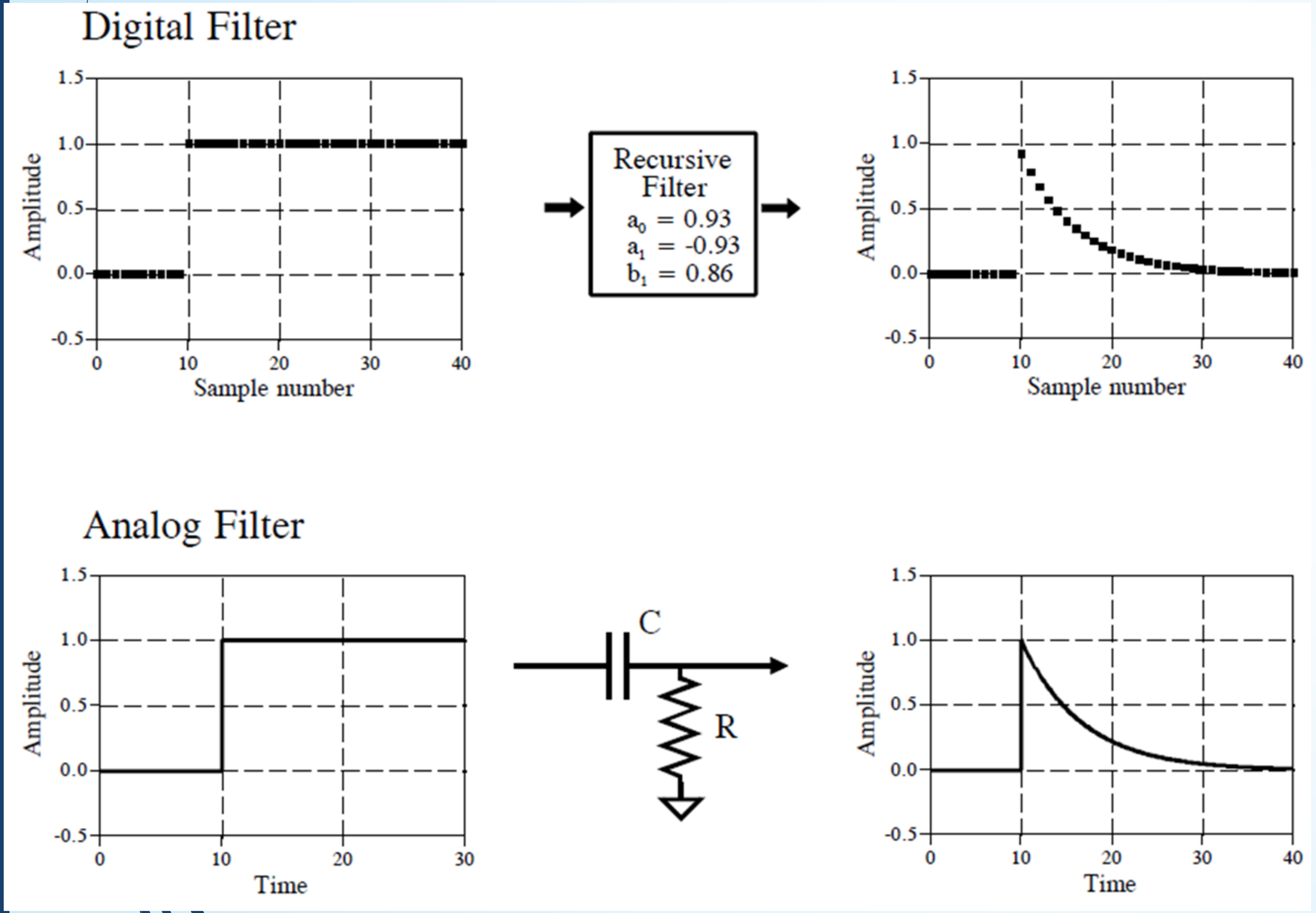


Figure shows an example of what is called a **single pole low-pass filter**. This recursive filter uses just two coefficients, $a_0 = 0.15$ and $b_1 = 0.85$. For this example, the input signal is a step function. As you should expect for a low-pass filter, the output is a smooth rise to the steady state level.

Digital recursive filters can mimic **analog filters** composed of resistors and capacitors. As shown in this example, a single pole low-pass recursive filter smoothes the edge of a step input, just as an electronic RC filter.

Single Pole Recursive Filters



The beauty of the recursive method is in its ability to create a **wide variety of responses** by changing only a few parameters. Figure shows a filter with three coefficients: $a_0 = 0.93$, $a_1 = -0.93$ and $b_1 = 0.86$ shown by the similar step responses, this digital filter mimics an electronic RC **high-pass filter**.

Single Pole Recursive Filters

You can use these **single pole recursive filters** to process digital signals just as you would use RC networks to process analog electronic signals.

This includes everything you would expect: DC removal, high-frequency noise suppression, wave shaping, smoothing, etc. They are **easy to program, fast to execute, and produce few surprises**. The coefficients are found from these simple equations:

$$\begin{aligned} a_0 &= 1 - x \\ b_1 &= x \end{aligned}$$

Single pole **low-pass filter**. The filter's response is controlled by the parameter, x, a value between zero and one.

$$\begin{aligned} a_0 &= (1 + x)/2 \\ a_1 &= -(1 + x)/2 \\ b_1 &= x \end{aligned}$$

Single pole **high-pass filter**.

Single Pole Recursive Filters

The characteristics of these filters are controlled by the parameter, $\boldsymbol{x}$, a value between **zero** and **one**.

Physically, $\boldsymbol{x}$ is the amount of ***decay*** between adjacent samples. For instance, $\boldsymbol{x}$ is 0.86 in above Figure, meaning that the value of each sample in the output signal is 0.86 the value of the sample before it. **The higher the value of x, the slower the decay.**

The filter becomes ***unstable*** if x is made **greater than one**. That is, any nonzero value on the input will make the output increase until an **overflow** occurs.

Single Pole Recursive Filters

The value for $\boldsymbol{x}$ can be directly specified, or found from the desired ***time constant*** of the filter. Just as *R*×C is the number of seconds it takes an RC circuit to **decay to 36.8% of its final value**, $\boldsymbol{d}$ is the number of samples it takes for a recursive filter to decay to this same level:

$$x = e^{-1/d}$$

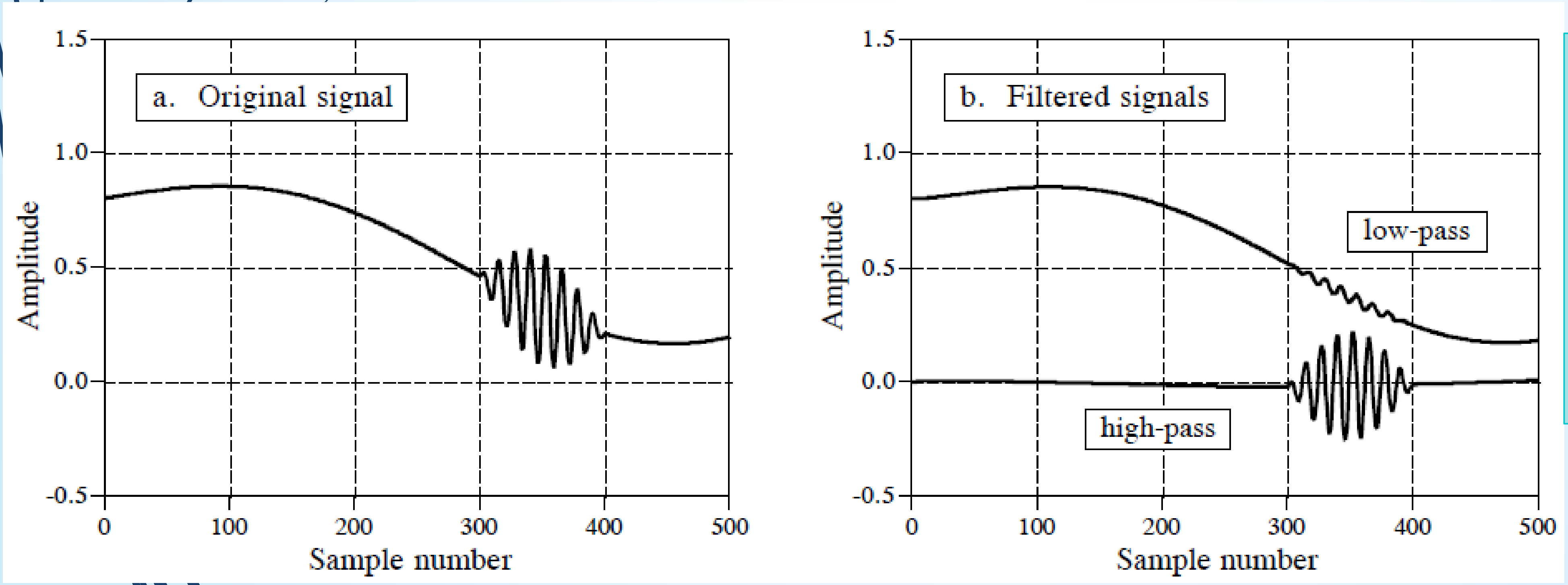
For instance, a sample-to-sample decay of $x = 0.86$ corresponds to a time constant of $d = 6.63$ samples (as shown in Figure above). There is also a fixed relationship between $\boldsymbol{x}$ and the -3dB ***cutoff frequency***, ***f*** , of the digital filter:

$$x = e^{-2\pi f_C}$$

This provides three ways to find the "$\boldsymbol{a}$" and "$\boldsymbol{b}$" coefficients, starting with the **time constant**, the **cutoff frequency**, or just directly picking $\boldsymbol{x}$.

Single Pole Recursive Filters

Figure shows an example of using single pole recursive filters. In (a), the original signal is a smooth curve, except a burst of a high frequency sine wave. Figure (b) shows the signal after passing through **low-pass** and **high-pass** filters. The signals have been separated fairly well, but not perfectly, just as if simple RC circuits were used on an analog signal.

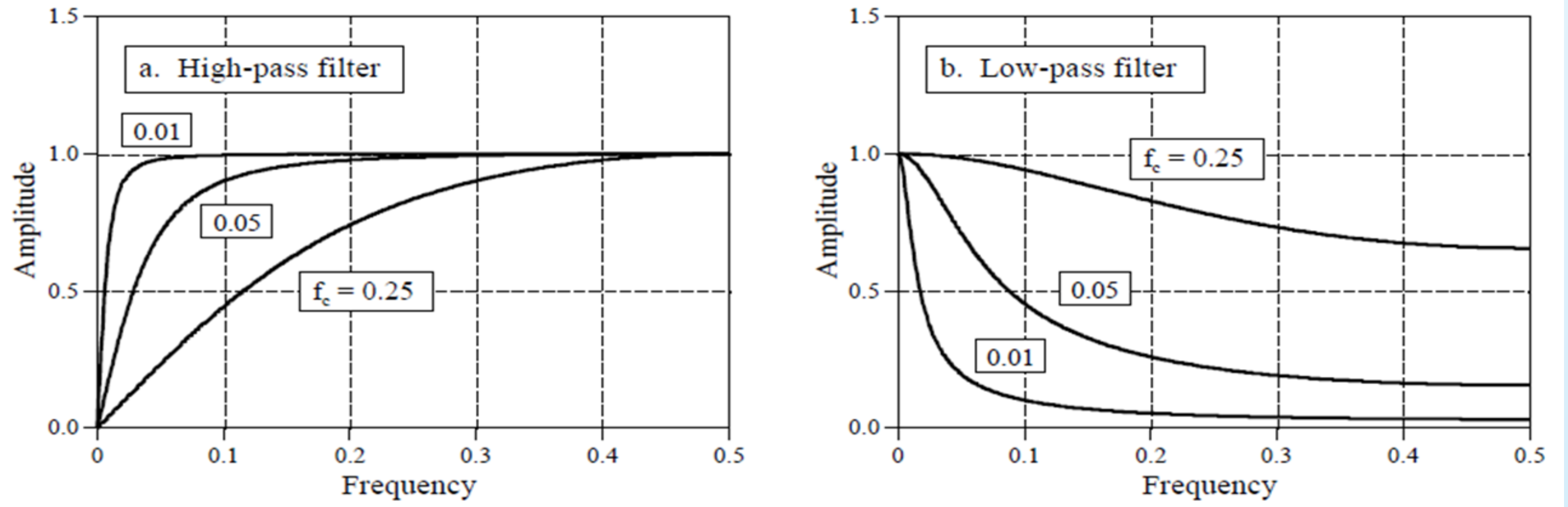


The low-pass filter uses $x = 0.95$, while the high-pass filter is for $x = 0.86$.

Single Pole Recursive Filters

Figure shows the frequency responses of various single pole recursive filters. These curves are obtained by passing a delta function through the filter to find the filter's impulse response. The FFT is then used to convert the impulse response into the frequency response.

In principle, the impulse response is **infinitely long**; however, it decays below the single precision roundoff noise after about 15 to 20 time constants. For example, when the time constant of the filter is $d = 6.63$ samples, the impulse response can be contained in about 128 samples.



Single Pole Recursive Filters

Single pole recursive filters have **little ability to separate one band of frequencies from another**. In other words, **they perform well in the time domain, and poorly in the frequency domain**. The frequency response can be **improved** slightly by cascading several stages. This can be accomplished in two ways.

1. The signal can be passed through the filter several times.
2. The z-transform can be used to find the recursion coefficients that combine the cascade into a single stage.

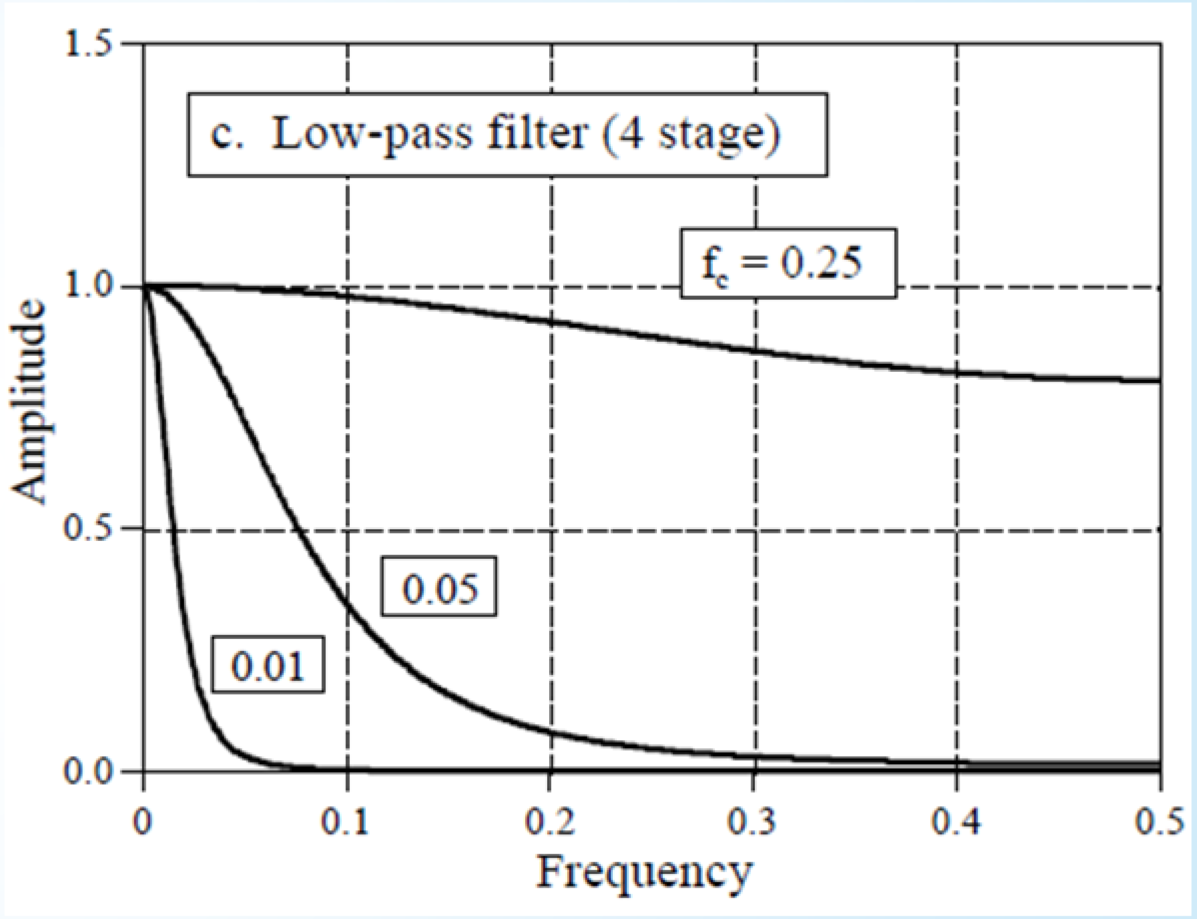
Both ways work and are commonly used.

Single Pole Recursive Filters

The frequency response of recursive filters is not always what you expect. For example, the $f_c = 0.25$ curve in (c) is quite useless.

Figure (c) shows the frequency response of a **cascade of four low-pass filters**.

Although the **stopband attenuation is significantly improved**, the roll-off is still terrible.



Four stage Low-pass Recursive Filters

The **four stage low-pass filter** is comparable to the Blackman and Gaussian filters, but with a much **faster** execution speed. The design equations for a four stage low-pass filter are:

$$\begin{aligned} a_0 &= (1-x)^4 \\ b_1 &= 4x \\ b_2 &= -6x^2 \\ b_3 &= 4x^3 \\ b_4 &= -x^4 \end{aligned}$$

Narrow-band Filters

A common need in electronics and DSP is to **isolate a narrow band** of frequencies from a wider bandwidth signal.

For example, you may want to eliminate 60 hertz interference in an instrumentation system, or isolate the signaling tones in a telephone network. Two types of frequency responses are available: the ***band-pass*** and the ***band-reject*** (also called a **notch filter**).

Narrow-band Filters

Recursion coefficients of these filters are provided by the following equations:

Band-pass filter

$$a_0 = 1 - K$$
$$a_1 = 2(K - R)\cos(2\pi f)$$
$$a_2 = R^2 - K$$
$$b_1 = 2R\cos(2\pi f)$$
$$b_2 = -R^2$$

Band-reject filter

$$a_0 = K$$
$$a_1 = -2K\cos(2\pi f)$$
$$a_2 = K$$
$$b_1 = 2R\cos(2\pi f)$$
$$b_2 = -R^2$$

where:

$$K = \frac{1 - 2R\cos(2\pi f) + R^2}{2 - 2\cos(2\pi f)}$$

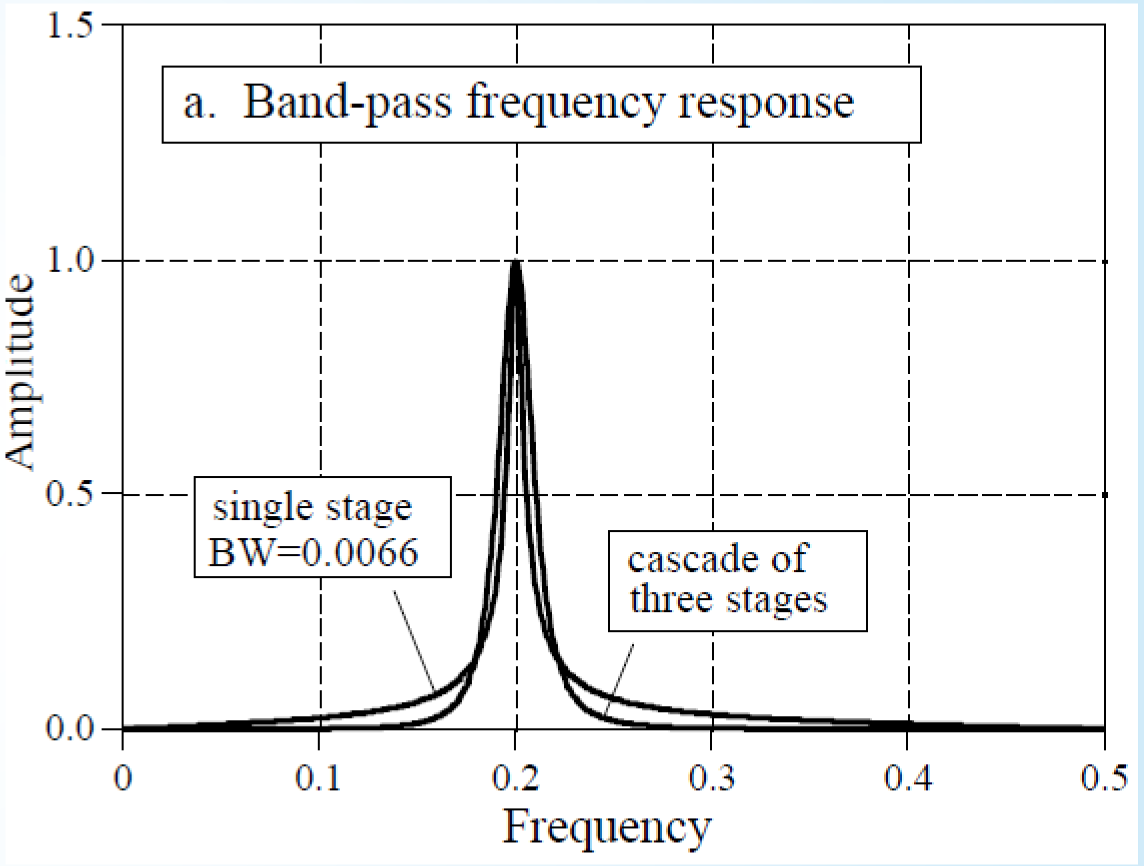
$$R = 1 - 3BW$$

To use these equations, first select the cetner frequency, ***f***, and the bandwidth, ***BW*** (measured at an amplitude of 0.707). Both of these are expressed as a fraction of the sampling rate, and therefore in the range of 0 to 0.5.

Next, calculate ***R***, and then ***K***, and then the recursion coefficients.

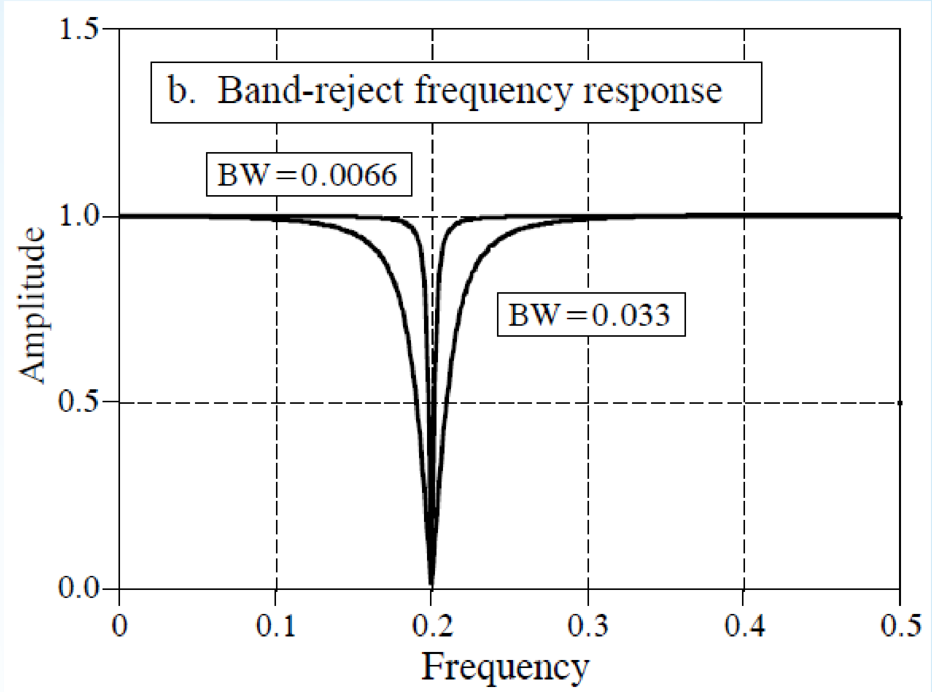
Band-pass Filters

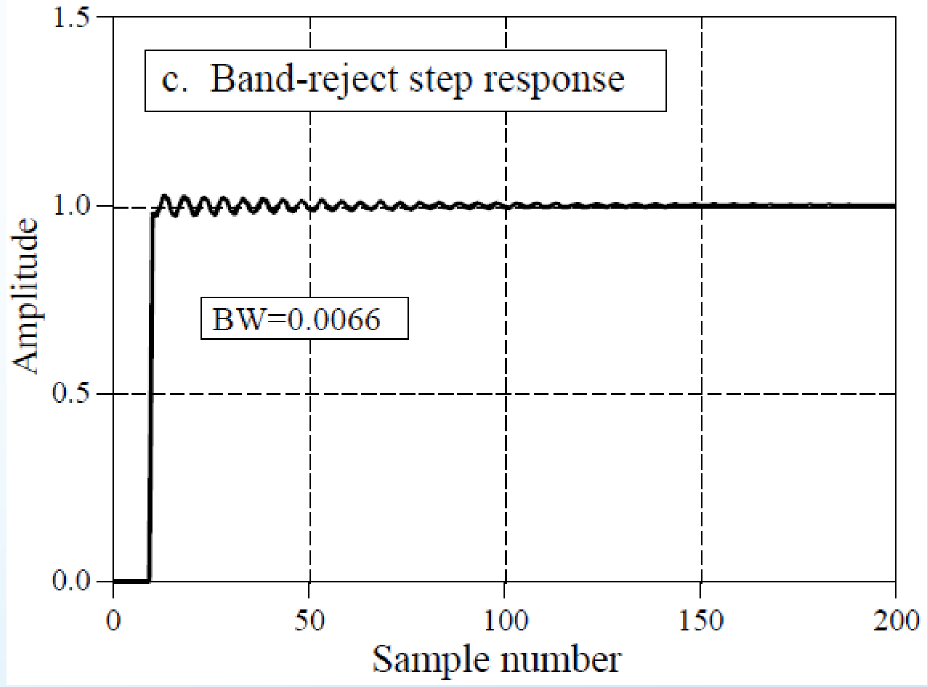
As shown in (a), the **band-pass** filter has relatively large ***tails*** extending from the main peak. This can be improved by cascading several stages. Since the design equations are quite long, it is simpler to implement this cascade by filtering the signal several times, rather than trying to find the coefficients needed for a single filter.



Band-reject Filters

Figure (b) shows examples of the **band-reject filter**. The narrowest bandwidth that can be obtain with single precision is about 0.0003 of the sampling frequency. Figure (c) shows the **step response** of the **band-reject filter**. There is noticeable overshoot and ringing, but its amplitude is quite small. This allows the filter to remove narrowband interference (60 Hz and the like) with only a minor distortion to the time domain waveform.

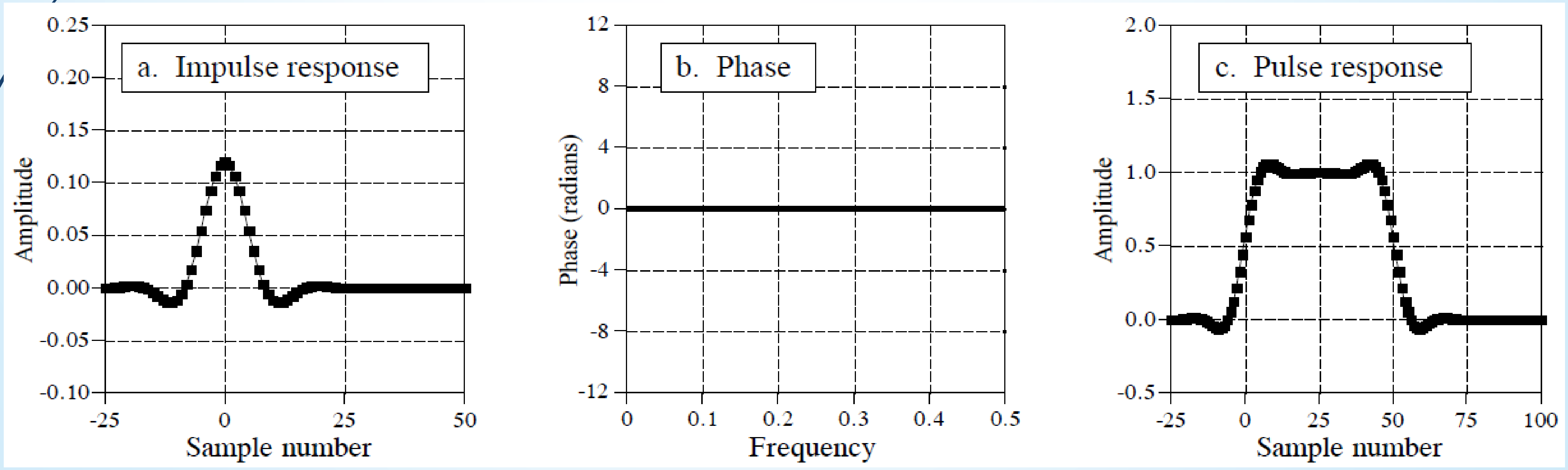




Phase Response: zero phase filter

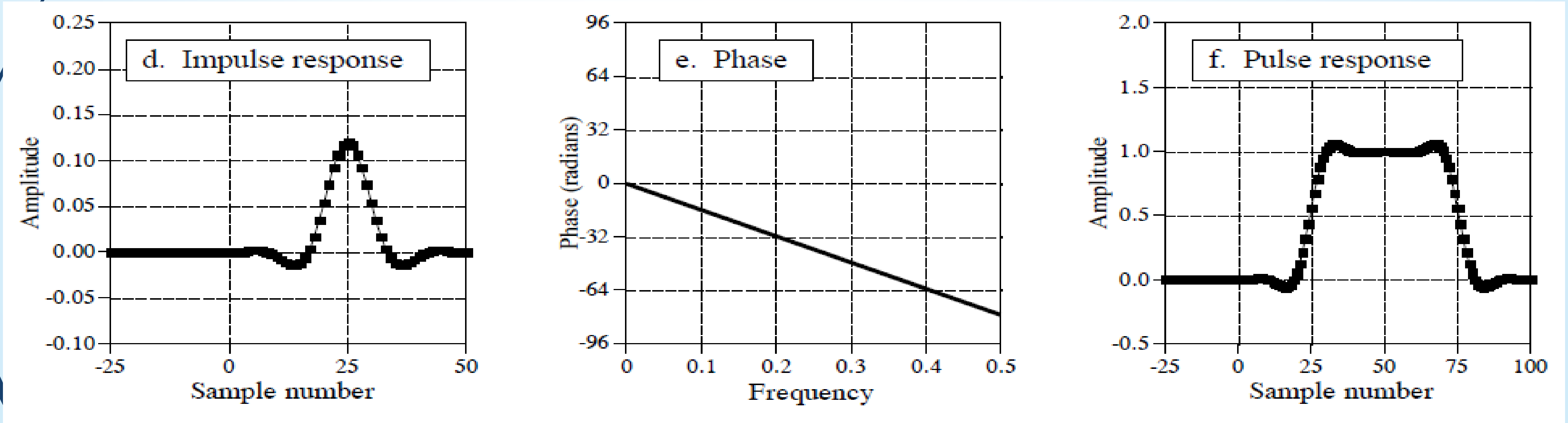
There are three types of ***phase response*** that a filter can have: **zero phase**, **linear phase**, and **nonlinear phase**.

The ***zero phase* filter** is characterized by an impulse response that is symmetrical around sample zero. The actual shape doesn't matter, only that the negative numbered samples are a mirror image of the positive numbered samples. When the Fourier transform is taken of this symmetrical waveform, the phase will be entirely zero, as shown in (b).



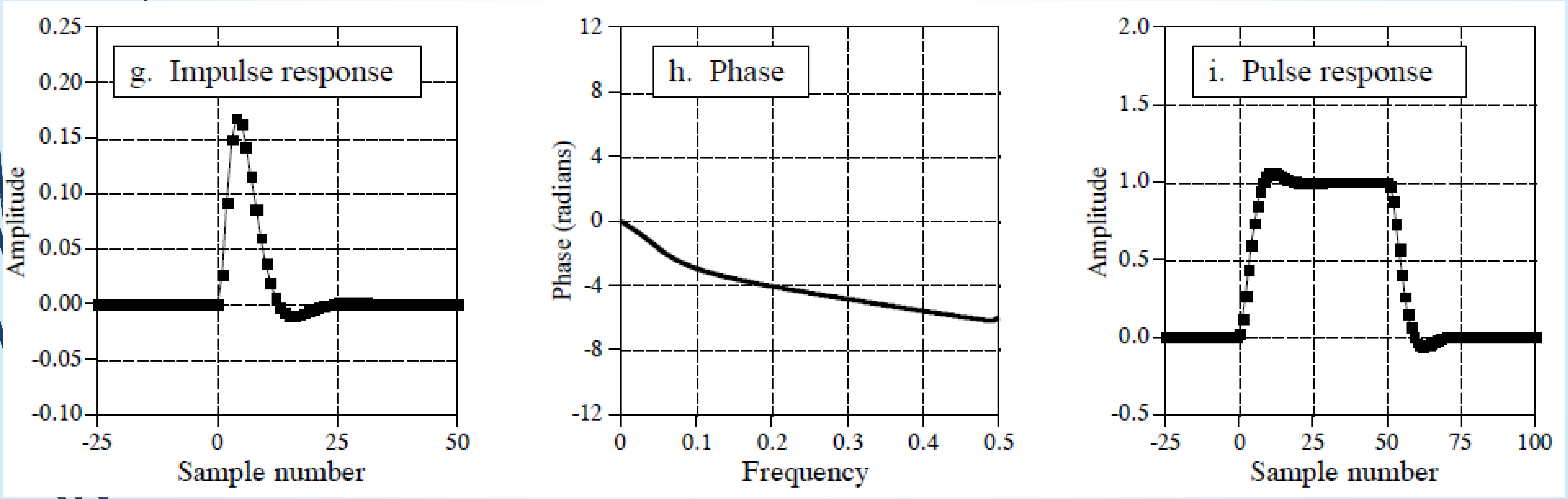
Phase Response: linear phase Filter

The impulse response of **linear phase filter** in (d) is identical to that shown in (a), except it has been shifted to use only positive numbered samples. The impulse response is still symmetrical between the left and right; however, the location of symmetry has been shifted from zero. This shift results in the phase, (e), being **a straight line: linear** *phase*. The slope of this straight line is directly proportional to the **amount of the shift**. Since the shift in the impulse response does nothing but produce an identical shift in the output signal, the **linear phase filter is equivalent to the zero phase filter** for most purposes.



Phase Response: nonlinear phase Filter

Figure (g) shows an impulse response that is ***not* symmetrical** between the left and right. Correspondingly, the phase, (h), is *not* a straight line. In other words, it has a **nonlinear phase**.



Phase Response

Why does anyone care if the phase is linear or not? These are the pulse responses of each of the three filters.

The **pulse response** is nothing more than a **positive going step response** followed by a negative going step response. The pulse response is used here because it displays what happens to both the rising and falling edges in a signal. Here is the important part: zero and linear phase filters have left and right edges that look the ***same***, while nonlinear phase filters have left and right edges that look ***different***. Many applications cannot tolerate the left and right edges looking different.

One example is in video processing. Can you imagine turning on your TV to find the left ear of your favorite actor looking different from his right ear?

Phase Response

It is easy to make an FIR (finite impulse response) filter have a linear phase. This is because the impulse response (filter kernel) is directly ***specified*** in the design process. **Making the filter kernel have left-right symmetry is all that is required**.

This is not the case with IIR (recursive) filters, since the **recursion coefficients are what is specified, not the impulse response**.

The impulse response of a recursive filter **is *not* symmetrical** between the left and right, and therefore has a **nonlinear phase**.

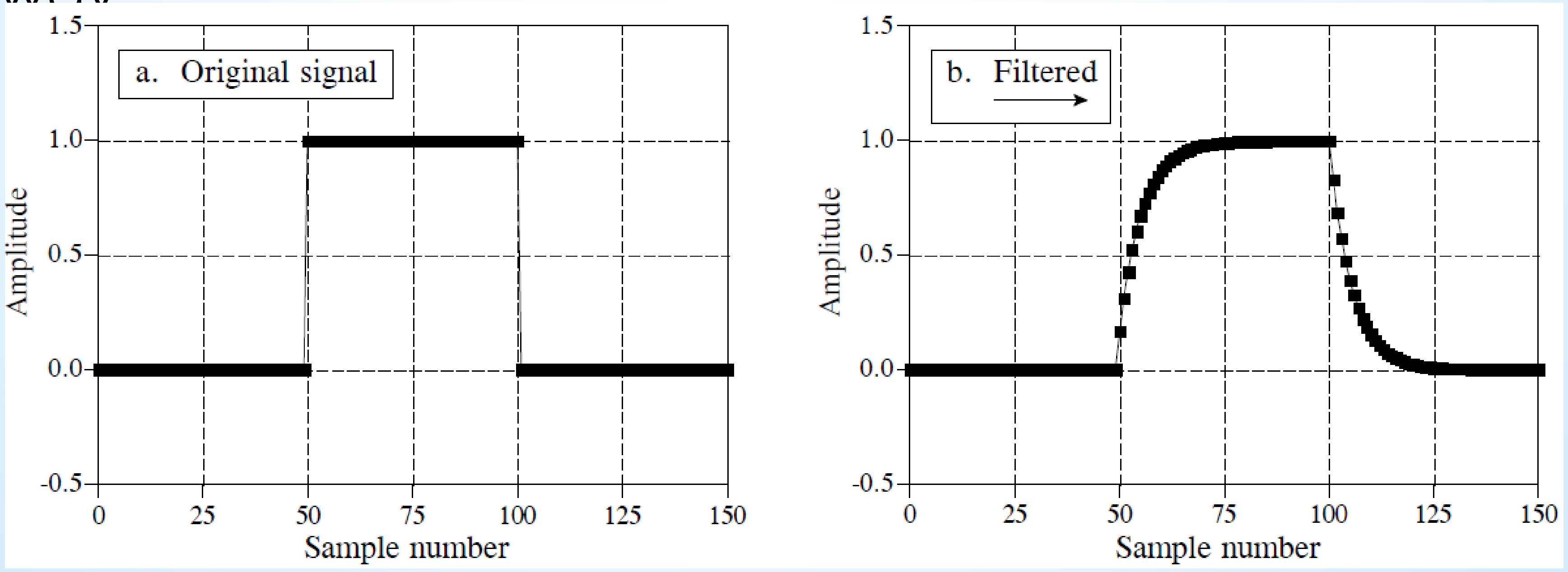
Phase Response

Analog electronic circuits have this same problem with the phase response. Imagine a circuit composed of resistors and capacitors sitting on your desk. If the input has always been zero, the output will also have always been zero. When an impulse is applied to the input, the capacitors quickly charge to some value and then begin to exponentially decay through the resistors. The impulse response (i.e., the output signal) is a combination of these various decaying exponentials. The impulse response ***cannot*** be symmetrical, because the output was zero before the impulse, and the exponential decay never quite reaches a value of zero again. Analog filter designers attack this problem with the **Bessel filter.** The Bessel filter is designed to have as linear phase as possible; however, it is far below the performance of digital filters.

The ability to provide an *exact* linear phase is a clear advantage of digital filters.

Phase Response

There is a simple way to modify recursive filters to obtain a ***zero phase***. Figure shows an example of how this works. The input signal to be filtered is shown in (a). Figure (b) shows the signal after it has been filtered by a single pole low-pass filter. Since this is a **nonlinear phase filter**, the left and right edges do not look the same; they are inverted versions of each other. As previously described, this recursive filter is implemented by starting at sample 0 and working toward sample 150, calculating each sample along the way.



Phase Response

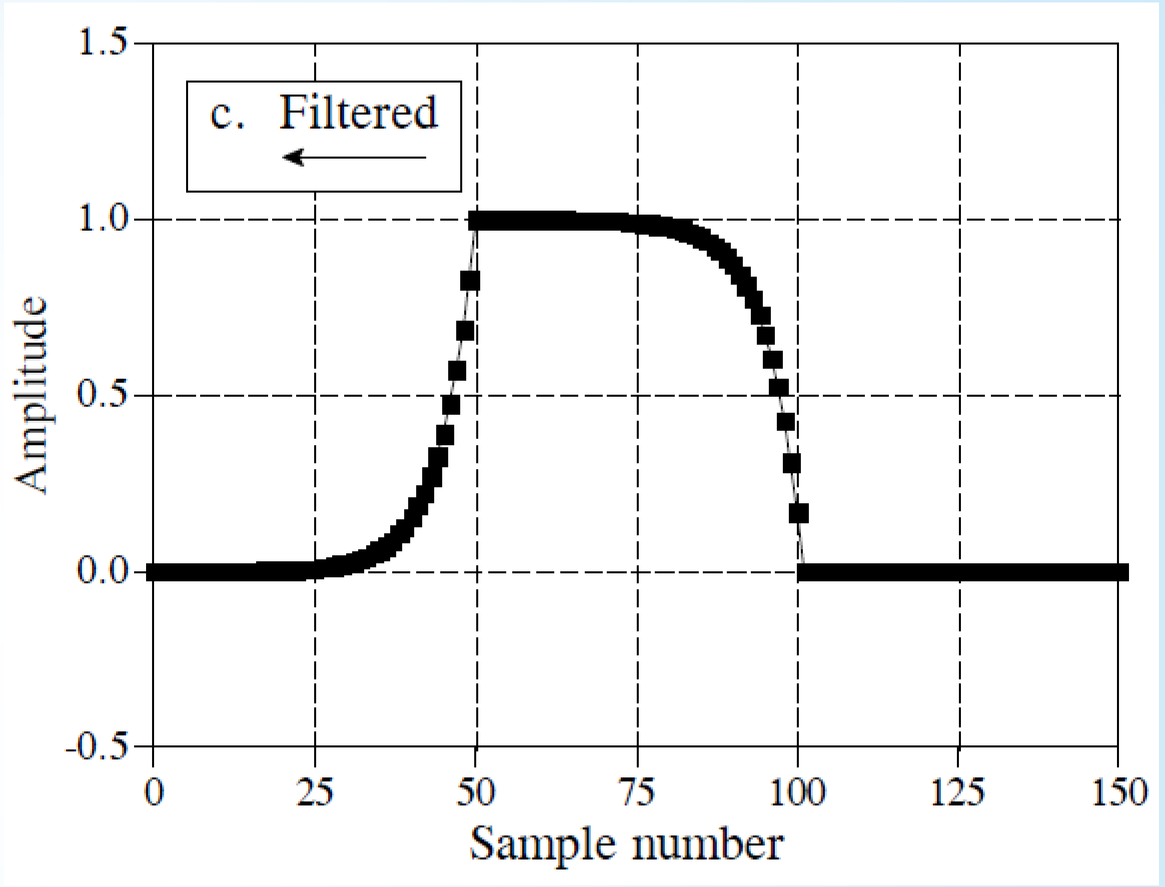
Now, suppose that instead of moving from sample 0 toward sample 150, we start at sample 150 and move toward sample 0. In other words, each sample in the output signal is calculated from input and output samples to the ***right*** of the sample being worked on. This means that the **recursion equation** is changed to:

$$\begin{aligned} y[n] = \; & a_0 x[n] + a_1 x[n+1] + a_2 x[n+2] + a_3 x[n+3] + \cdots \\ & + b_1 y[n+1] + b_2 y[n+2] + b_3 y[n+3] + \cdots \end{aligned}$$

Phase Response

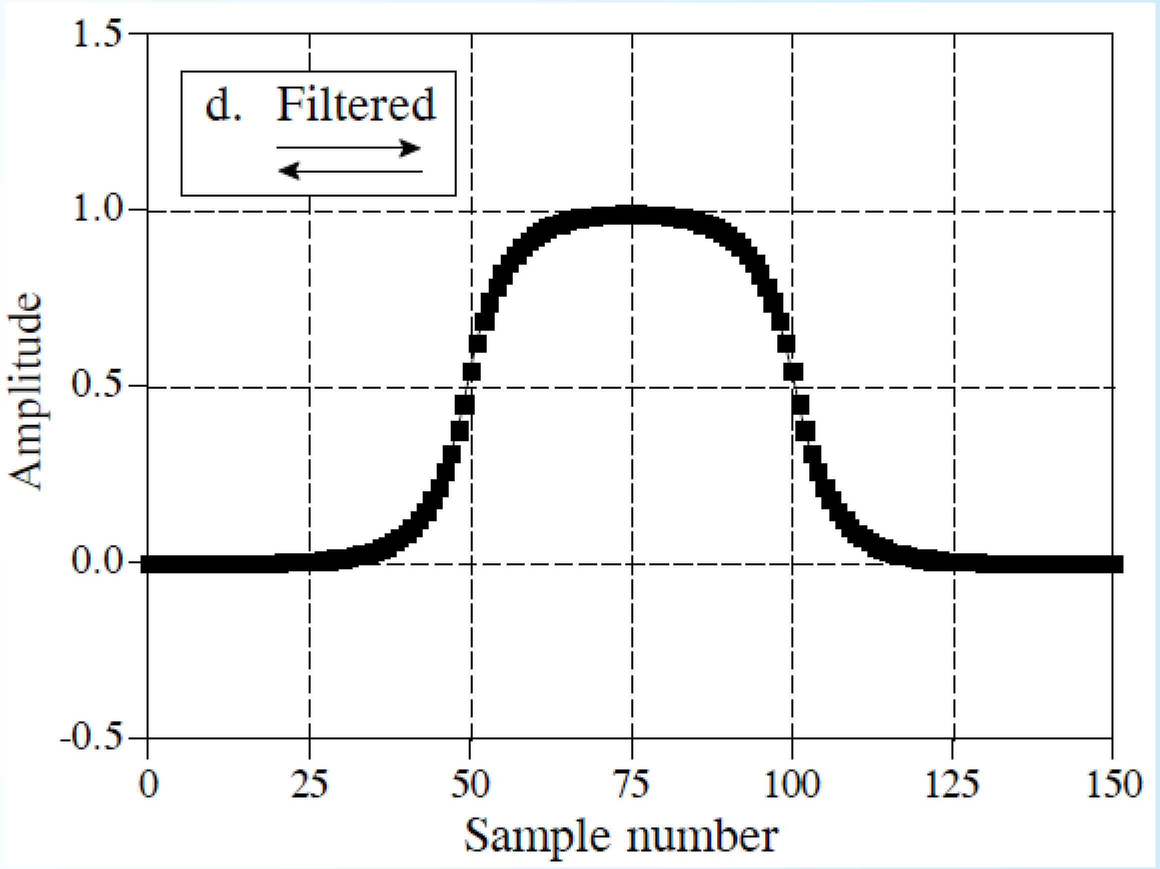
Figure (c) shows the result of this **reverse filtering**. This is analogous to passing an analog signal through an electronic RC circuit while running time ***backwards***. **!esrevinu eht pu-wercs nac lasrever emit - noituaC**

Filtering in the reverse direction does not produce any benefit in itself; the filtered signal still has left and right edges that do not look alike.



Phase Response: Bidirectional filtering

The magic happens when forward and reverse filtering are *combined*. Figure (d) results from filtering the signal in the forward direction and then filtering again in the reverse direction. Voila! This produces a ***zero phase* recursive** filter. In fact, *any* recursive filter can be converted to zero phase with this **bidirectional filtering** technique. The only penalty for this improved performance is a factor of two in **execution time** and **program complexity**.



Phase Response

How do you find the impulse and frequency responses of the **overall** filter?

Magnitude of the frequency response is the **same** for each direction, while the phases are **opposite in sign**.

When the two directions are combined, the **magnitude becomes *squared***, while the **phase cancels to *zero***.

In the time domain, this corresponds to **convolving the original impulse response with a left-for-right flipped version of itself**. For instance, the impulse response of a single pole low-pass filter is a one-sided exponential. The impulse response of the corresponding bidirectional filter is a one-sided exponential that decays to the right, convolved with a one-sided exponential that decays to the left. Going through the mathematics, this turns out to be a double-sided exponential that decays both to the left and right, with the same decay constant as the original filter.