7 Basics of Fourier Analysis and Filtering

Overview

When you record the EEG and construct averaged ERP waveforms, you represent the data as a voltage value at each time point. However, it is also possible to use *Fourier analysis* to represent EEG and ERPs in terms of the sum of a set of oscillating voltages at various frequencies. This frequency-based representation is essential for understanding how filters work, and it is also useful for understanding time–frequency analyses. Unfortunately, many people misunderstand what frequency-based representations really tell us, and this also leads to the misuse of filters and time–frequency analyses. The goal of this chapter is to provide a basic overview of Fourier analysis and filtering. This overview will be sufficient for most conventional ERP studies. If you are interested in doing more sophisticated frequency-based analyses, you can find more detailed information in online chapters 11 and 12.

I begin this chapter with a simple overview of Fourier analysis. I then describe the conventional approach to filtering EEG and ERP data, which is based on Fourier analysis. This is followed by a mathematically equivalent approach to filtering that is based entirely on time, which is often more appropriate for EEG and ERP data. I then describe how filters can distort your data, and then I finish with some concrete recommendations for how and when you should filter your data. I use very little math in presenting this material (nothing beyond arithmetic and averaging). My goal is to give you the conceptual background to understand frequency-based representations and filtering, along with some practical advice about how you should filter your data.

Filtering is essential, but it can lead to severe distortions if applied incorrectly. I will show examples of these distortions near the end of the chapter, and then I'll provide you with some simple recommendations that will help you avoid them. You may want to deviate from these recommendations, but I would strongly recommend following them until you have read and fully understand online chapters 11 and 12 (which provide the math).

You may already know a bit about Fourier analysis and filtering, so you might be tempted to skip part or all of this chapter. However, even though this chapter focuses on the basics, you should at least skim it, because you will probably discover some important principles. For

example, if you think that the presence of power at 18 Hz in a Fourier analysis means that a brain oscillation was present at 18 Hz, you should definitely read this chapter. Similarly, you should read this chapter if you don't know that a high-pass filter with a half-amplitude cutoff of 0.5 Hz can produce artifactual peaks in your data or if you don't know why high-pass filters should ordinarily be applied to the continuous EEG rather than to averaged ERPs.

Basics of Fourier Analysis

Transforming ERPs into Frequency-Domain Representations

Figure 7.1A shows an ERP waveform that is contaminated by 60-Hz line noise. You can tell that it's 60 Hz because there are six peaks in this noise every 100 ms (and therefore 60 peaks every 1000 ms). This waveform is equivalent to the sum of the "clean" ERP waveform and a 60-Hz sinusoidal oscillation (see the glossary if you need a reminder about sine waves). Fourier analysis, named after the French mathematician Joseph Fourier, provides a way of determining the amplitude and phase of the 60-Hz oscillation in this complex waveform. This then allows us to design a filter that can remove the 60-Hz oscillation, leaving us with the clean ERP waveform.

Although it's fairly obvious that a 60-Hz oscillation is present in the waveform shown in figure 7.1A, we can actually decompose the entire waveform into sine waves. Joseph Fourier proved that any waveform—no matter how complex—can be re-created by summing together a set of sine waves of various frequencies, amplitudes, and phases. This is truly a stunning and counter-intuitive idea: The waveform shown in figure 7.1A certainly doesn't look like it could be re-created by summing together a bunch of sine waves. Well, it can, and this forms the basis of many widely used techniques in mathematics, engineering, and science.

A mathematical procedure called the *Fourier transform* is used to determine the frequencies, amplitudes, and phases of the sine waves that need to be added together to re-create a given waveform.[1] For example, figure 7.1B shows the result of applying the Fourier transform to the ERP waveform in figure 7.1A. This is called transforming the data from the *time domain* (which just means that the *X* axis is time) into the *frequency domain* (which just means that the *X* axis is frequency). The frequency-domain representation shown in figure 7.1B shows the amplitude needed for each frequency of sine wave in order to reconstruct the ERP waveform shown in figure 7.1A. There is also a phase value for each frequency (not shown here). If you took a set of sine waves of the frequencies and amplitudes shown in figure 7.1B (with the appropriate phases), and you added them together, you would get exactly the same waveform shown in figure 7.1A. If you did this, you would be performing the *inverse Fourier transform*, which is the mathematical procedure that transforms a frequency-domain representation back into a time-domain representation.

Note that the amplitude in figure 7.1B is higher at 60 Hz than at the surrounding frequencies. This tells us that we would need to include a fairly large sine wave at 60 Hz to reconstruct the ERP waveform, which is consistent with the fact that the waveform obviously contains a 60-Hz

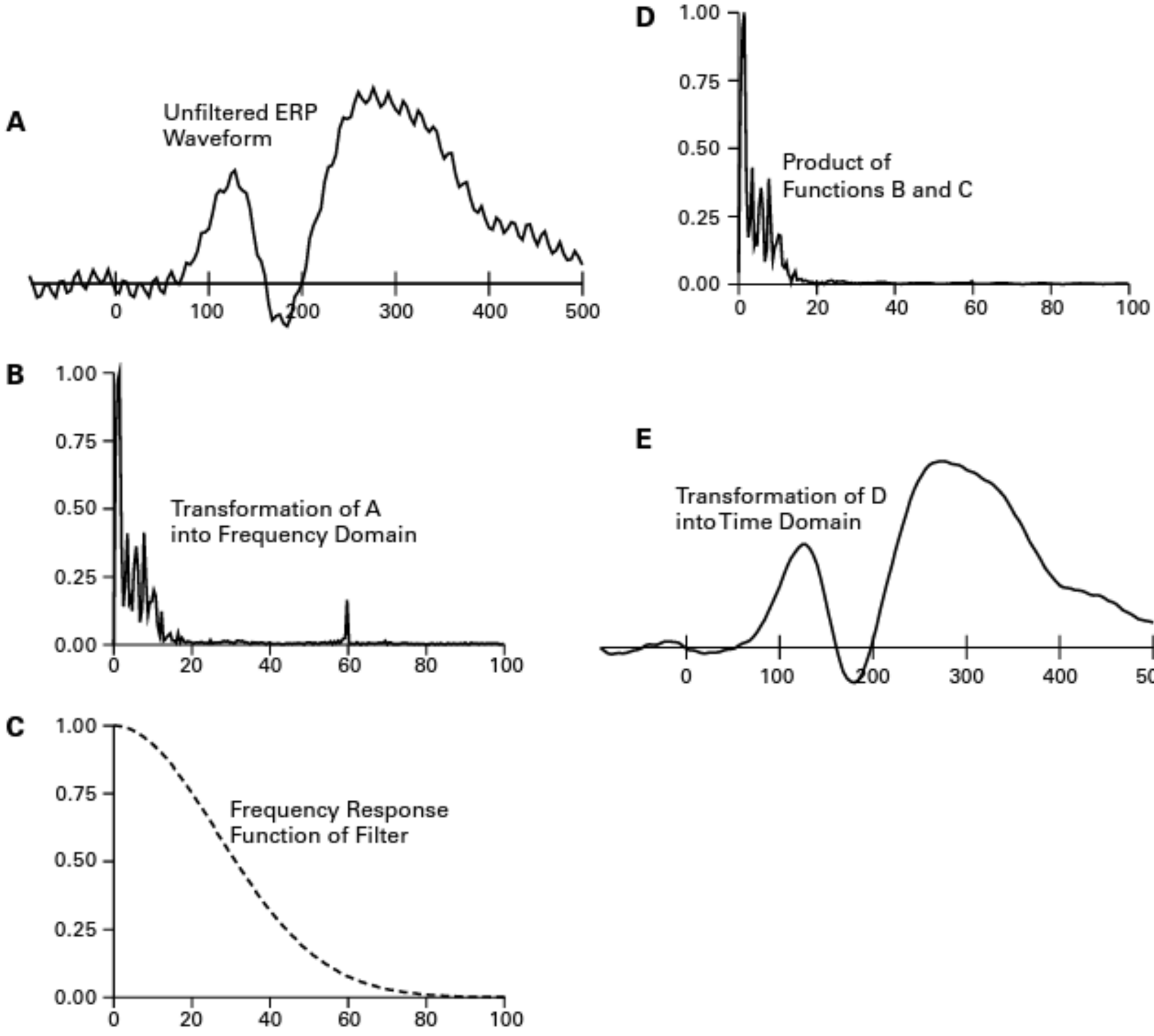

Figure 7.1
Frequency-based conceptualization of filtering. (A) Unfiltered ERP waveform, contaminated by substantial noise at 60 Hz. (B) Transformation of the waveform from panel A into the frequency domain by means of the Fourier transform. Note the clear peak at 60 Hz. (C) Frequency response function of a low-pass filter that can be used to attenuate high frequencies, including the 60-Hz noise. (D) Product of the waveforms in panels B and C; for each frequency point, the magnitude in panel B is multiplied by the magnitude in panel C. Note that panel D is nearly identical to panel B in the low-frequency range, but it falls to zero at high frequencies. (E) Transformation of the waveform from panel D back into the time domain by means of the inverse Fourier transform. The resulting waveform closely resembles the original ERP waveform in panel A, except for the absence of the 60-Hz noise. Note that the phase portion of the frequency-domain plots has been omitted here for the sake of simplicity. Also, *Y*-axis units have been omitted to avoid the mathematical details, except for panel C, in which the *Y* axis represents the gain of the filter (the scaling factor for each frequency).

oscillation. There are also many frequencies under 15 Hz that have a relatively high amplitude in figure 7.1B, but it is less obvious that we need these frequencies. This is the "magic" of Fourier analysis: Even though it isn't visually apparent that we need these frequencies, these are in fact the frequencies we would need to reconstruct the ERP waveform from sine waves.

People often use power rather than amplitude to represent the strength of each sine wave. Power is simply amplitude squared, and amplitude is simply the square root of power. Using power instead of amplitude has some advantages, but it's really the same information expressed on a different scale.

You need to be aware of one small caveat about applying the Fourier analysis to ERP waveforms. Specifically, sine waves have an infinite duration, whereas ERP waveforms do not. Consequently, if you reconstructed the original waveform by summing together a bunch of infinite-duration sine waves, the ERP waveform would repeat over and over again, both backward and forward in time. That is, the 600-ms waveform that we see from –100 to +500 ms would repeat again from 500 to 1100 ms, from 1100 to 1700 ms, and so forth. It would also repeat from –700 to –100 ms, from –1300 ms to –700 ms, and so forth. This isn't really a problem in practice, and it doesn't mean that something is wrong with Fourier analysis. After all, we've given the Fourier transform no information about the voltages that were present outside this narrow time window, so we can't expect it to come up with anything sensible beyond this window. If we gave it an infinite-duration waveform as the input (e.g., by making the prestimulus interval extend to negative infinity and the end of the ERP extend to positive infinity), the reconstruction of this waveform from sine waves would not show this repetition. Nonetheless, this is a warning sign that we need to be careful about how we interpret the results of a Fourier analysis. We'll return to this issue soon.

You may have heard people talk about the "sine and cosine components" of a Fourier transform. These are sometimes called the "real and imaginary components." These are just alternative ways of talking about amplitude and phase, and there is nothing "imaginary" about them. If you are interested in understanding these terms, see the online supplement to chapter 7.

The Fourier Transform of a Simple Square Wave

Figure 7.2 shows the application of the Fourier transform to a simpler time-domain waveform, a repeating square wave. You might think that it would be impossible to re-create a square wave—which consists of straight lines and sharp corners—from a set of smoothly varying sine waves. However, this example shows that you can, in fact, reconstruct the square wave from sine waves, as long as you're willing to add together enough sine waves. Unlike the Fourier transform of the ERP waveform in figure 7.1, which requires sine waves of many consecutive frequencies, it turns out that the Fourier transform of a square wave requires a distinct subset of possible frequencies. Specifically, if the square wave repeats at a frequency of F times per second, the sine waves needed to reconstruct the square wave are at frequencies of F, 3F, 5F, 7F, and so forth, without any of the in-between frequencies. These frequencies are called the *fundamental frequency* (F) and the *odd harmonics* (3F, 5F, 7F, etc.).

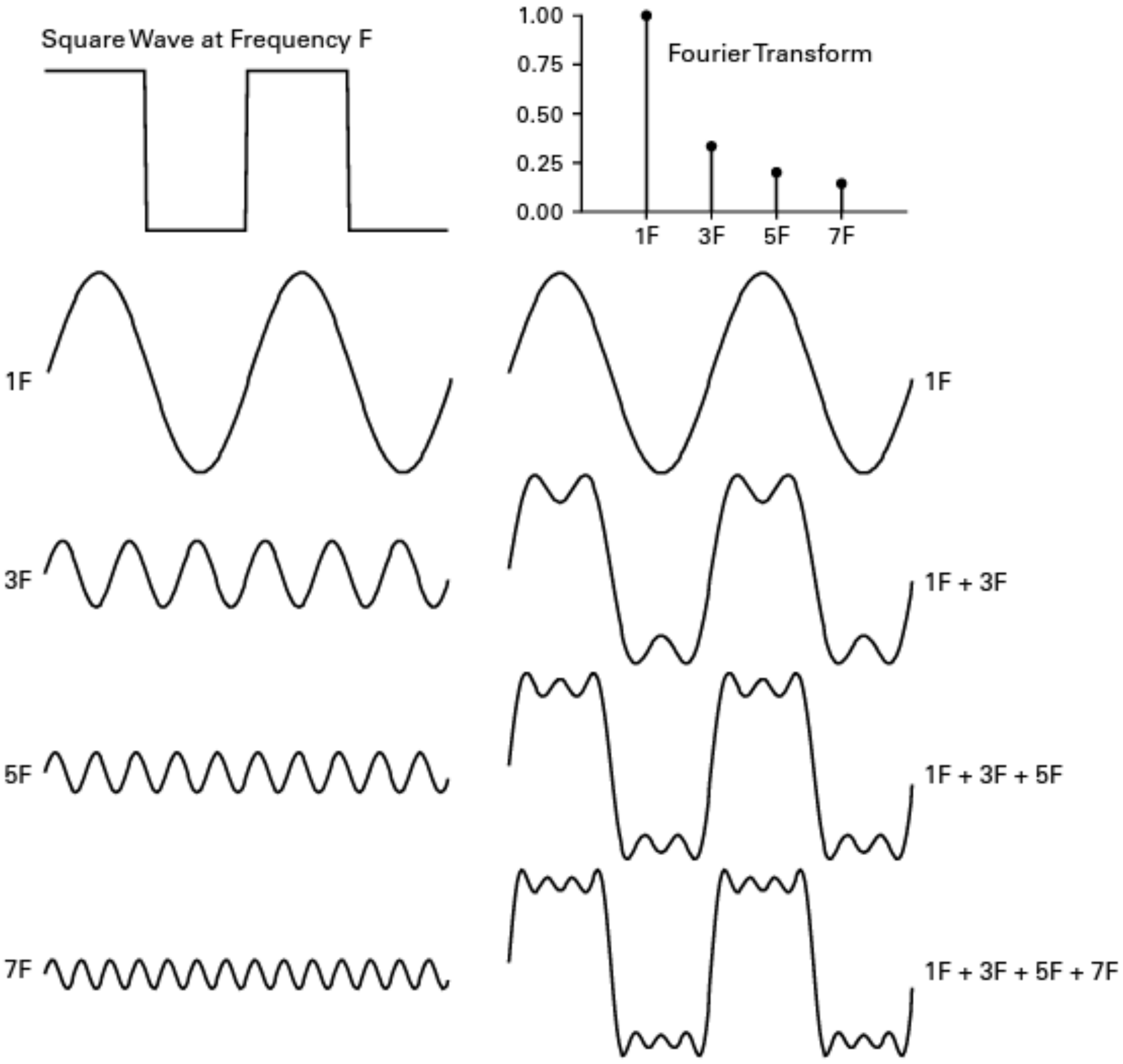

Figure 7.2
Example of the application of Fourier analysis to a repeating square wave.

If you add the 1F and 3F sine waves (see 1F + 3F in figure 7.2), the result looks a little more like a square wave, but it still lacks straight lines and sharp corners. Once you add 1F, 3F, 5F, and 7F, however, the result starts to approximate the square wave. I hope you can imagine that we would get closer and closer to a perfect square wave by adding more and more of the odd harmonics together. It takes an infinite number of these odd harmonics to perfectly reconstruct a square wave, but you can get extremely close with a relatively modest number.

What a Fourier Transform Really Means

People often make a fundamental error in interpreting Fourier transforms (and related processes, such as time–frequency analyses). The problem is that they mistake *mathematical* equivalence

for *physiological* equivalence. The essence of Fourier analysis is that any waveform can be reconstructed by adding together a set of sine waves. This does not mean, however, that the waveform actually *consists* of a set of sine waves. For example, the Fourier transform shown in figure 7.1B has some power at 18 Hz, and this means that we would need to include an 18-Hz sine wave if we wanted to add together a set of sine waves to re-create the ERP waveform shown in figure 7.1A. It does not mean that the brain was generating an 18-Hz sine wave when it produced this ERP waveform. This is just like saying that a $1 bill is equal in value to 100 pennies. You can exchange a $1 bill for 100 pennies, but the bill is not physically made of 100 pennies. You can also exchange a $1 bill for nine dimes and two nickels. These are all mathematically equivalent, but they are not physically equivalent. This brings us to a key concept that you should commit to memory:

Fundamental principle of frequency-based analyses Power at a given frequency does not mean that the brain was oscillating at that frequency.

When I run ERP boot camps, I always ask the participants whether they find this fundamental principle surprising. At least half of them say yes. And this is consistent with my experience reading journal articles that discuss frequency-based analyses. That is, at least half of the papers I read implicitly assume that power at a given frequency means that the brain was actually oscillating at this frequency. For example, the paper might say something like "We observed a neural oscillation between 25 and 60 Hz that differed across conditions." This conclusion usually goes well beyond the actual data, because they did not actually *observe* an oscillation; they merely saw power in a particular frequency range in a Fourier-based analysis. In essence, these papers are *assuming* that the data consist of the sum of a set of sine waves when they apply the frequency-based analysis technique, and the conclusion is simply a restatement of this assumption. No matter whether the data contain sine wave oscillations or not, there will always be power at some frequencies. Power at a frequency in a Fourier-based (or wavelet-based) analysis is therefore a mathematical inevitability, not evidence for physiological oscillations.

After I describe this fundamental principle at an ERP Boot Camp, the first question I am always asked is this: "If power at a given frequency doesn't mean that the brain was oscillating at that frequency, then what does it mean?" It just means that we would need some power at that frequency if we were trying to reconstruct the observed data by summing together a set of sine waves. The summing of sine waves is something the *experimenter* is doing to reconstruct the waveforms, not necessarily something the *brain* is doing when it is generating the waveforms. The brain may or may not be oscillating at this frequency, and Fourier analysis cannot tell us whether the brain is actually oscillating.

Of course, a true oscillation at a given frequency will lead to power in that frequency in a Fourier transform. For example, the ERP waveform in figure 7.1A contains a 60-Hz oscillation that was induced by oscillating electrical activity in the environment. Consequently, power at 60 Hz in the Fourier transform shown in figure 7.1B really was a result of a 60-Hz oscillation

in the data. Similarly, you can often see alpha band oscillations (approximately 10 Hz) in the raw EEG. The alpha oscillations appear to reflect, at least in part, recurrent connections between the cortex and the thalamus that produce a loop, and brain activity oscillates at 10 cycles per second as signals travel around in this loop (Schreckenberger et al., 2004; Klimesch, Sauseng, & Hanslmayr, 2007). The alpha activity typically leads to high power in a narrow band of frequencies around 10 Hz in a Fourier transform of the EEG. This is another case in which a true oscillation can be measured by examining the power at a specific frequency in a Fourier transform. But the power at 18 Hz in the Fourier transform of the ERP waveform shown in figure 7.1 probably does not reflect a neural oscillation at 18 cycles per second. The 18-Hz power reflects a mathematical equivalence, not a physiological equivalence.

There is an asymmetry here: A neural oscillation at a given frequency will virtually always produce power at that frequency in a Fourier analysis, but power at a given frequency in a Fourier analysis does not entail that the brain was oscillating at that frequency.

The next question I'm asked at the ERP Boot Camp is this: "How can you tell when power at a given frequency reflects a true oscillation at that frequency?" In most cases, strong power in a narrow range of frequencies in a Fourier analysis reflects a true oscillation in that range of frequencies. For example, there is strong power from 58 to 60 Hz in figure 7.1B, with very low power from 40 to 58 Hz and from 62 to 100 Hz, and this reflects the 60-Hz oscillation that you can see in the ERP waveform in figure 7.1A. In contrast, the power at 18 Hz is not much different from the power at 16 Hz or 20 Hz, and this activity probably doesn't reflect a true oscillation. Thus, if you see a narrow band of high power, it is probably an oscillation, but if it is a broad range of frequencies, it is probably not an oscillation. In particular, a transient brain response (i.e., a short-duration, non-oscillating voltage deflection) usually contains power all the way down to 0 Hz. There are exceptions to this, but it's a good heuristic in most cases. The reasoning behind this heuristic is provided in online chapter 12.

If you're reading a paper that shows some kind of frequency-based analysis, and it doesn't show you what happens at the lower frequencies (e.g., <10 Hz), you should be skeptical of any conclusions the paper draws about oscillations. True oscillations may be present, but it's hard to be certain without seeing the low frequencies. For example, figure 7.3 shows a simulated time–frequency analysis. The *X* axis is time relative to stimulus onset, the *Y* axis is frequency, and the darkness of the shading represents change in power relative to baseline. Periods of gamma-band power (30–40 Hz) are present from approximately 100 to 300 ms and from approximately 400 to 800 ms. The gamma-band power from 400 to 800 ms is likely to reflect true neural oscillations, because it is isolated to a fairly narrow band of frequencies (30–40 Hz), with no power at lower frequencies. In contrast, the power from 100 to 300 ms extends down to the lowest frequency shown (20 Hz), making it impossible to know whether this is a real oscillation or whether it is instead some kind of transient, non-oscillating response. I won't name names, but I have seen many papers in which activity like that shown from 100 to 300 ms was interpreted as an oscillation, even though it is quite likely that it reflected a transient, non-oscillating brain response.

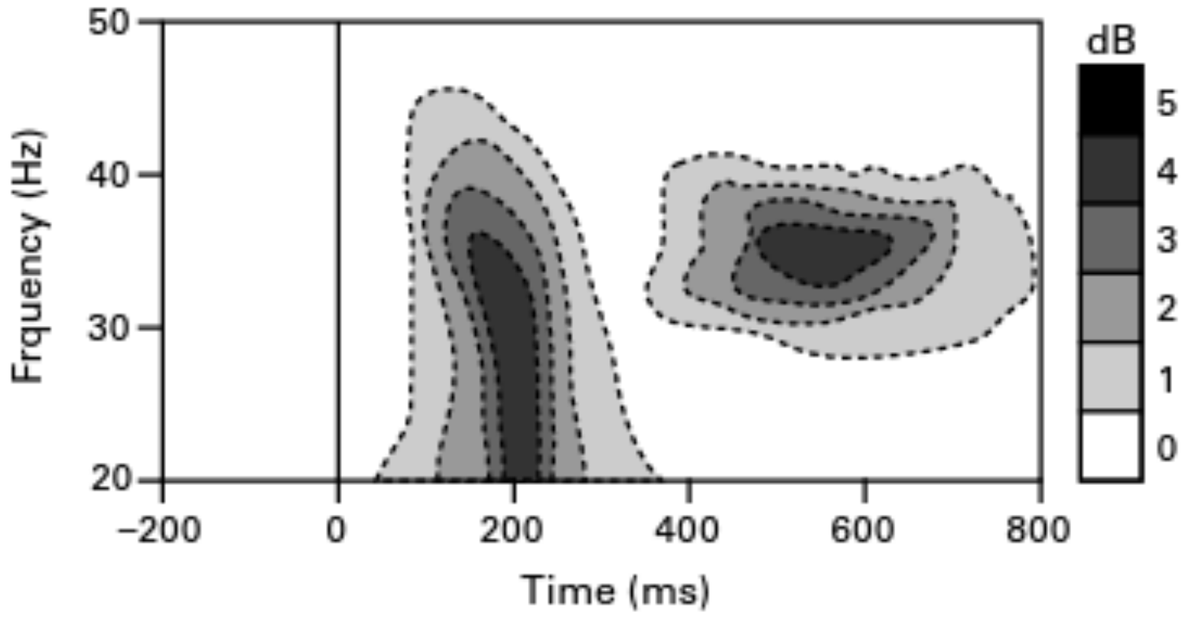

Figure 7.3
Simulated time–frequency data. The *X* axis shows time, the *Y* axis shows frequency, and the darkness indicates the change in power relative to the prestimulus baseline (in units of decibels, dB).

The final question that I get from ERP Boot Camp participants is usually this: "Why does it matter whether something is truly an oscillation?" The answer is that the presence of an oscillation in a specific frequency band has implications for the kind of neural circuitry that is producing the brain activity. For example, gamma-band oscillations are often assumed to involve short-range recurrent connections between fast-spiking, parvalbumin-expressing interneurons and pyramidal cells (see, e.g., Knoblich, Siegle, Pritchett, & Moore, 2010; Volman, Behrens, & Sejnowski, 2011). Moreover, oscillations are often a sign of neural synchrony, which may play an important role in representing and processing information in neural circuits (see, e.g., Singer, 1999). Thus, it can be useful to know if the brain is oscillating at a given frequency. Unfortunately, researchers often assume that they have detected an oscillation without very good evidence that the brain is indeed oscillating.

Basics of Filtering in the Frequency Domain

Now that I've explained the basics of Fourier analysis, it's time for you to learn the basics of filtering. Here I will explain the most common approach to filtering, which focuses on suppressing specific frequency bands. This approach treats ERPs as if they actually consist of the sum of a set of sine waves, which sometimes leads to interpretive errors. Nonetheless, the most common uses of filters involve reducing sources of noise that can be well approximated by sine waves, so there is considerable value to this approach. And once you understand this approach, you will be ready to understand the time-domain approach to filtering.

Why Are Filters Necessary?

A key message of this chapter is that filters can substantially distort your data. It is therefore useful to ask why it's worthwhile to use filters. There are two answers to this question, the first of which is related to the *Nyquist theorem* (see chapter 5). This theorem states that it is possible

to convert a continuous analog signal (like the EEG) into a set of discrete samples without losing any information, as long as the rate of digitization is more than twice as high as the highest frequency in the signal being digitized. This means that we can legitimately store the EEG signal as a set of discrete samples on a computer. However, this theorem also states that if the original signal contains frequencies that are more than twice as high as the digitization rate, the high frequencies will appear as artifactual low frequencies in the digitized data (this is called *aliasing*). Consequently, EEG recording systems use hardware-based *anti-aliasing* filters to suppress high frequencies, and these filters are generally set to eliminate frequencies at or above the *Nyquist frequency* (one-half of the sampling rate). For example, a typical cognitive ERP experiment might use a digitization rate of 250 Hz, and it would therefore be necessary to make sure that everything greater than or equal to 125 Hz is eliminated.

The second main goal of filtering is noise reduction, and this is considerably more complicated. The basic idea is that the EEG consists of a signal plus some noise, and some of the noise can be suppressed simply by attenuating certain frequencies. For example, most of the relevant portion of the ERP waveform in a typical cognitive neuroscience experiment consists of frequencies between approximately 0.1 Hz and 30 Hz, whereas EMG activity primarily consists of frequencies above 100 Hz; consequently, suppressing frequencies above 100 Hz will greatly reduce the EMG activity while producing very little change to the EEG signal. However, if the frequency content of the signal and the noise are similar, it is difficult to suppress the noise without significantly distorting the signal. For example, alpha waves can provide a significant source of noise, but because they are within the range of frequencies of the ERP, it is difficult to filter them without significantly distorting the ERP waveform.

In addition to suppressing high frequencies, filters are also used in most experiments to attenuate very low frequencies, which typically arise from the electrodes and the skin rather than from the brain (as was discussed in chapter 5). It is usually a good idea to remove these slow voltage shifts by filtering frequencies lower than approximately 0.1 Hz. This is especially important when recordings are obtained from patients or from children, because head and body movements are one common cause of these sustained shifts in voltage. You can reduce these slow voltage shifts even more by filtering with a higher cutoff, such as 0.5 or 1.0 Hz, but these frequencies also contribute in an important way to the ERP waveform. Filtering with such a high cutoff may cause substantial distortion of the ERP waveforms and may decrease the statistical power (as described later in this chapter).

I should also note that filters are often useful for helping you visually inspect your average ERP waveforms. If these waveforms contain a lot of high-frequency activity, this will make it difficult for you (and the people reviewing your journal submissions) to see the differences between conditions or groups. For this reason, it can be useful to filter out the high-frequency noise before plotting the data.

Filters are usually described in terms of their ability to suppress or pass various different frequencies. The most common classes of filters are (1) *low-pass filters*, which attenuate high frequencies and pass low frequencies; (2) *high-pass filters*, which attenuate low frequencies and

pass high frequencies; (3) *band-pass filters*, which attenuate both high and low frequencies, passing only an intermediate range of frequencies; and (4) *notch filters*, which attenuate some narrow band of frequencies (e.g., 60 Hz) and pass everything else. Personally, I find it very confusing to describe a filter in terms of the frequencies that it passes, and I would find it much more natural to refer to a *high-block* filter than a *low-pass* filter. But this is the convention, and we need to live with it. Note that a band-pass filter is equivalent to filtering once with a low-pass filter and then filtering a second time with a high-pass filter (or vice versa).

Filtering as Multiplication in the Frequency Domain

The effects of a filter are usually expressed by the filter's *frequency response function,* which describes how each frequency is influenced by the filter. Usually, we focus on how the filter changes the amplitude at each frequency, which is determined by the *gain* at each frequency. The gain is a multiplication factor; each frequency in the input of the filter is multiplied by the gain for that frequency. A gain of 1.0 means that the amplitude of that frequency is unchanged by the filter. A gain of 0.25 means that the amplitude is multiplied by 0.25 and is therefore reduced by 75%. A gain of 1.5 means that the amplitude is multiplied by 1.5 and is therefore increased by 50%. In the typical filters used with EEG/ERP data, the gain is between 0 and 1 for each frequency.

The frequency response function of a typical low-pass filter is shown in figure 7.1C. It has a gain near 1 for frequencies below 10 Hz, so these frequencies are not influenced much by the filter. It has a gain near zero for frequencies above 80 Hz, so these frequencies are almost completely eliminated by the filter. There is a gradual decline in gain between 5 and 80 Hz, and these frequencies are partly suppressed and partly passed by the filter. The gain is approximately 0.10 at 60 Hz, so 90% of the 60-Hz noise will be suppressed by the filter.

The actual filtering process is quite simple when conceptualized in this manner. The original ERP waveform (panel A of figure 7.1) is transformed into the frequency domain with the Fourier transform (panel B). The amplitude of each frequency in the frequency-domain representation of the data is then multiplied by the corresponding gain value in the frequency response function (panel C), leading to a filtered version of the frequency-domain data (panel D). The filtered frequency-domain data are then transformed back into the time domain with the inverse Fourier transform, giving us a filtered version of the time-domain ERP waveform (panel E).

If you compare the original and filtered versions of the frequency-domain data (panel B versus panel D), you can see that the pattern of amplitudes in the low frequencies (under 10 Hz) is very similar for the filtered and unfiltered data. This is because the gain values are near 1.0 for these frequencies. However, the spike at 60 Hz in the unfiltered data (panel B) is nearly eliminated in the filtered data (panel D), because the gain at 60 Hz is approximately 0.10. Consequently, the rapid up-and-down noise oscillations that you can see in the unfiltered ERP waveform (panel A) are largely eliminated in the filtered ERP waveform (panel E).

You can use any function you want for the frequency response function. For example, you could use a function that has a gain of 1 for every frequency between 0 and 50 Hz and a gain

of 0 for every frequency above 50 Hz. You might think this would be a much better frequency response function, because you wouldn't have the partial attenuation of intermediate frequencies that is produced by the filter in figure 7.1. However, as I will describe in more detail later, this sort of very sudden transition in the frequency response function can have negative side effects.

Filtering Twice

What happens if you filter the same data twice? For example, you might use a hardware low-pass filter during data acquisition to avoid aliasing, and then you might want to apply a low-pass filter offline in software to further reduce high-frequency noise in the data. You can understand how this would work in terms of the sequence of operations shown in figure 7.1. Filtering twice just repeats this sequence of operations twice.

However, there is a much more convenient way to think about filtering twice. Specifically, you can just multiply the frequency response functions of the two filters together, and this will give you the frequency response function that results from applying both filters. This is illustrated in figure 7.4, which shows the frequency response functions of two filters, labeled "filter A" and "filter B." Filtering the data twice, once with filter A and then once with filter B, is equivalent to filtering the data once with a frequency response function that is created by simply multiplying the frequency response functions for the two individual filters. This is true even if one of the filters is implemented in hardware and the other is implemented in software.

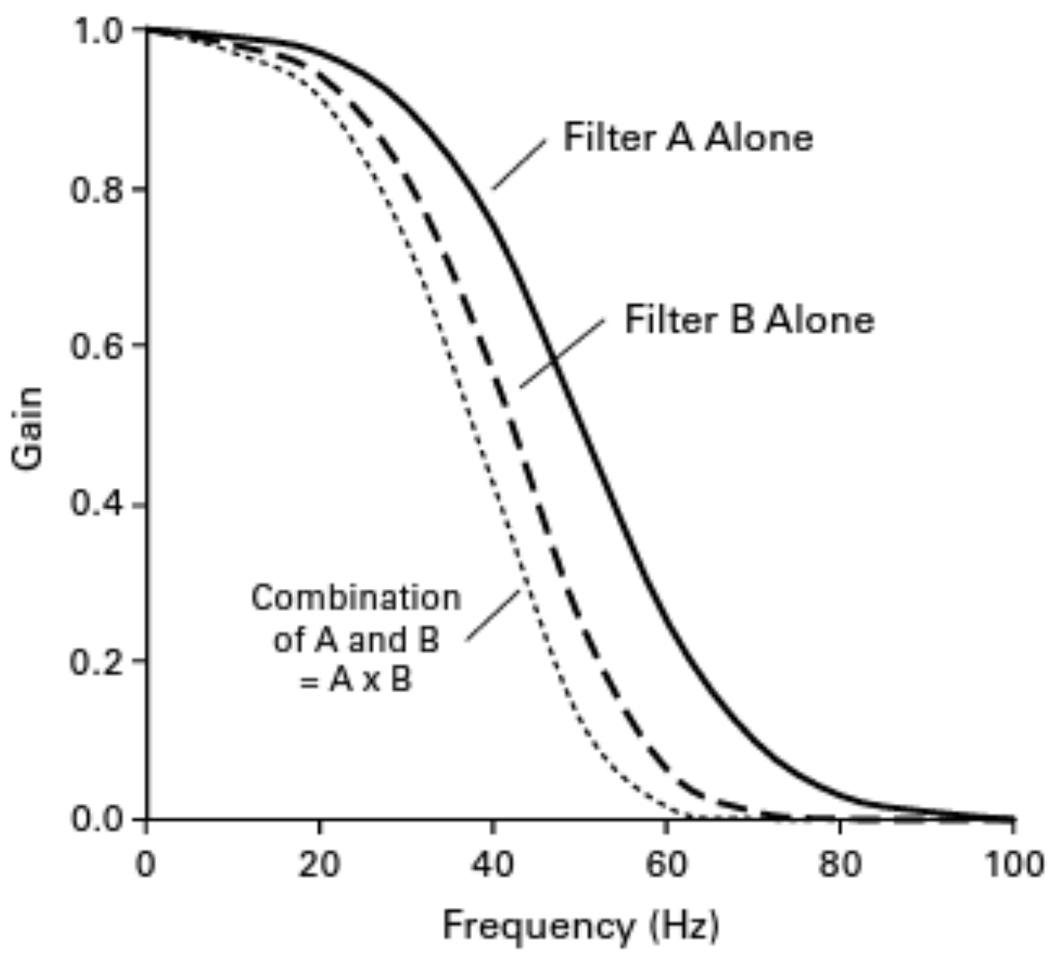

Figure 7.4
Frequency response functions of two filters, A and B, along with the effective frequency response function that would be obtained by filtering the data with both filters in succession. Filtering the data with filter A and then filtering with filter B is equivalent to filtering once with a frequency response function that is the product of the frequency response functions of filter A and filter B. That is, the frequency response function of the combination of the two filters is computed by multiplying the gains of the two individual functions at each frequency.

You might think that filtering a second time with exactly the same filter wouldn't change your data. After all, if you've already eliminated a frequency, how could filtering a second time change that frequency? But filtering a second time will change your data. Imagine, for example, that we applied filter A from figure 7.4 twice. This filter has a gain of approximately 0.80 at 38 Hz. The first time we filter the data, we would remove 20% of the signal at 38 Hz. The second time, we would remove 20% of the remaining signal at 38 Hz. Similarly, filter A has a gain of approximately 0.71 at 41 Hz. If we filter twice with this filter, the gain of the double filtering will be 0.5 at 41 Hz (because $0.71 \times 0.71 = 0.5$). Thus, whereas filter A has a half-amplitude cutoff at approximately 50 Hz, filtering twice with this filter gives us a half-amplitude cutoff at approximately 41 Hz.

Phase Shifts

When we use the Fourier transform to create a frequency-domain representation, as in figure 7.1B, there is a phase as well as an amplitude for each frequency (although this is frequently omitted from figures, including the figures in this book). Similarly, a frequency response function specifies a phase shift as well as a gain for each frequency. In most offline (software) filters, the phase shift is zero for every frequency, so you don't need to worry about phase with these filters. Hardware filters, however, always have a phase shift, which moves the ERP waveform rightward in a complicated manner. The amount of shift is negligible for most antialiasing filters, which have a relatively high cutoff frequency, so you don't usually need to worry about the phase shift. However, if you use hardware filters with a relatively low cutoff frequency (e.g., under 50 Hz), you might produce a meaningful latency shift. This is one reason why it is best to do most of the filtering offline, using filters that do not shift the phase or latency.

As I will explain in more detail later in the chapter, filters create a filtered value at a given time point by means of a weighted combination of the surrounding time points. Most offline filters do this in a symmetrical manner, giving equal weight to the previous and subsequent time points. As a result, they do not produce a phase shift. In contrast, hardware filters only "know" about the past time points, because the future time points haven't yet happened, so only previous voltages are used to create the filtered value at a given time point. The voltage at a given time point influences the value at subsequent time points, and this effectively "pushes" this value later in time, causing a phase shift. Such filters are called *causal* filters, because they follow the basic principle of causation that the past can influence the future but the future cannot influence the past.

Offline filters use both previous and subsequent voltages to compute the filtered value at a given time point. Thus, they violate the unidirectionality of time and are therefore called *noncausal* filters. Although it might sound like a bad idea to use something that does not obey a basic principle of causation, this is not usually a problem in practice. In almost all cases, you will want to use noncausal filters for your offline filtering to avoid latency shifts.

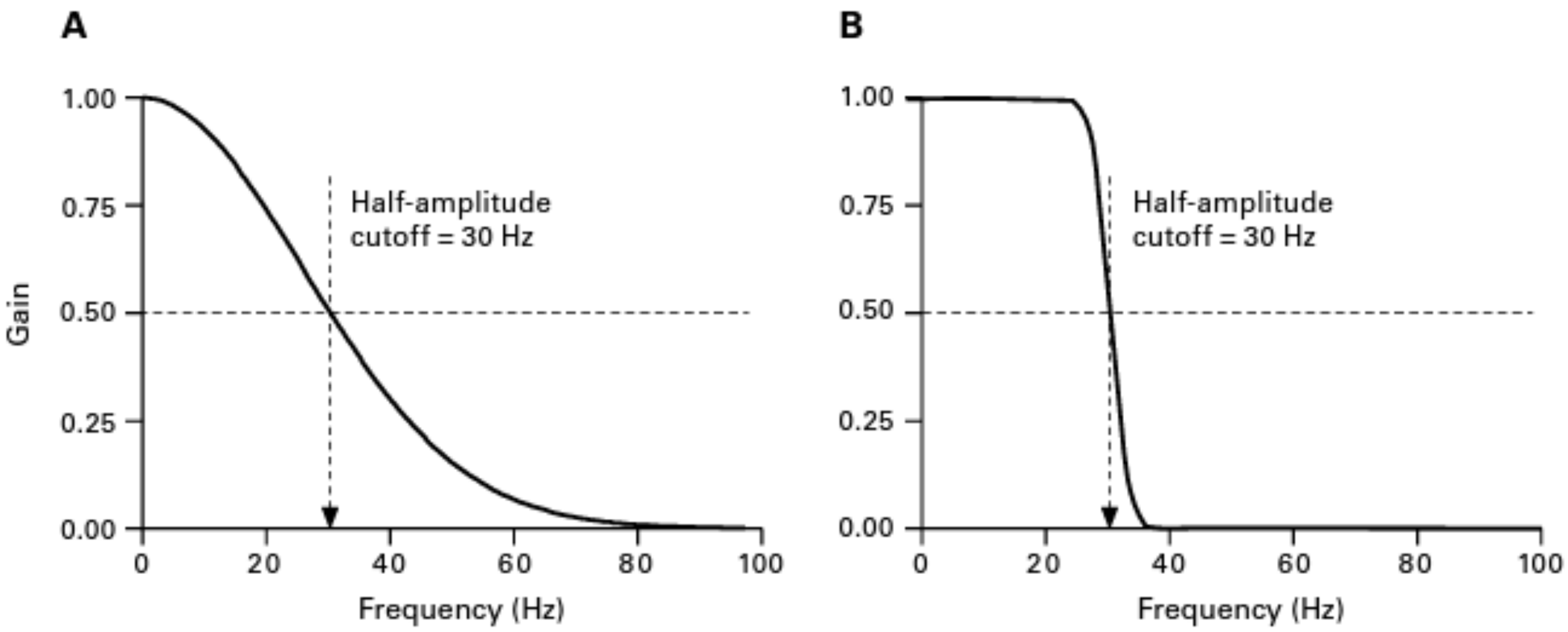

Figure 7.5
Examples of the frequency response functions of two filters that have the same half-amplitude cutoff (30 Hz) but with a gentle roll-off (A) or a steep roll-off (B).

Cutoff Frequency, Roll-off, and Slope

In ERP research, a filter is often described solely in terms of its *half-amplitude cutoff*, which is the frequency at which the amplitude is reduced by 50%. However, this does a poor job of describing the entire frequency response function. For example, figure 7.5 shows two frequency response functions that both have a half-amplitude cutoff at 30 Hz. The function in panel A falls off very gently between 10 and 80 Hz, whereas the function in panel B is very flat from 0 to 25 Hz, falls rapidly between 25 and 35 Hz, and is then very flat again.

The steepness of the decline in gain is called the *roll-off* of the filter, and it is typically quantified by the *slope* of the frequency response function at the cutoff frequency. The frequency response function in figure 7.5A has a shallow roll-off, whereas the frequency response function in figure 7.5B has a steep roll-off. The slope is usually specified in dB/octave, which is a logarithmic scale. A decrease of 3 dB is a 50% drop in power, and a change of 6 dB is a 50% drop in amplitude. An octave is a doubling of frequency. Thus, a filter with a half-amplitude cutoff at 30 Hz and a slope of 6 dB/octave has a 50% drop in amplitude (6 dB) between 20 and 40 Hz (one octave).[2]

Note that the cutoff frequency is sometimes specified in terms of power rather than amplitude. Whereas the half-amplitude cutoff is the point at which the gain is reduced by 6 dB, the half-power cutoff is the point at which the gain is reduced by only 3 dB. A half-power cutoff might be described like this: "The data were low-pass filtered (–3 dB at 30 Hz)." The amplitude is reduced by only 29% at the point where the power is reduced by 50%. If the slope of the filter is fairly gentle, the half-power cutoff might be quite different from the half-amplitude cutoff. For example, a filter with a slope of 12 dB/octave that had a half-power cutoff of 30 Hz would

Box 7.1
What Kind of Filter Was That?

I was an associate editor at *Cognitive, Affective, and Behavioral Neuroscience* for several years, and in that role I handled almost all of the ERP manuscripts that were submitted to the journal. In addition to making big-picture decisions about whether each submission was publishable, I made sure that every ERP paper in the journal met a set of minimal standards for describing the methods and results. Much to my surprise, I found that the vast majority of submitted manuscripts failed to provide a remotely adequate description of the filters that were applied. For example, there was often no mention of the hardware filter that was used to eliminate aliasing during data acquisition. In addition, most papers that mentioned filter cutoffs failed to specify whether the specified cutoffs were half-amplitude or half-power values. And most papers neglected to indicate the slope of the filter. In fact, one paper involved comparing experiments conducted in different laboratories, and it wasn't clear that the filters used for the two data sets were equivalent. The graduate student who wrote the paper had to learn a lot about filtering to satisfactorily revise the paper (which was eventually accepted).

I also heard a story about two labs that tried to replicate each other but kept failing, even though they thought they were processing their data in exactly the same way. They eventually discovered that the discrepancy occurred because one lab was specifying the cutoff frequency in terms of the half-amplitude point and the other was specifying the cutoff frequency in terms of the half-power point. Once they used equivalent filters, the discrepancy was eliminated.

The bottom line is that you need to describe your filters in enough detail so that the reader knows what you did to your data and can replicate your data processing procedures. This should be an obvious point, but most ERP researchers don't seem to realize how much filtering can influence the results. Don't make this mistake in your own papers. At a minimum, you should indicate the cutoff frequency, whether this was the half-amplitude or half-power point, and the slope of the roll-off. It also helps to describe the general class of filter (e.g., Gaussian, Bessel, Butterworth), but that is a more advanced topic that will be described in online chapter 12.

have a half-amplitude cutoff at approximately 46 Hz. That's a 50% higher cutoff frequency when expressed in terms of amplitude instead of power! See box 7.1 for a brief rant about the importance of fully specifying your filters when you write a paper for publication.

High-Pass Filters and the Time Constant

Up to this point, we have mainly considered low-pass filters. In most cognitive experiments, low-pass filters help to reduce induced electrical noise and EMG noise. High-pass filters are used to reduce slow changes in voltage caused by skin potentials and other gradual changes in the voltage offset, which can improve statistical power. An example of the application of a high-pass filter to a 30-s period of continuous EEG is provided in figure 7.6. The unfiltered data exhibit a clear upward drift over this time period, and this drift is eliminated by the filter, which had a half-amplitude cutoff of 0.1 Hz and a slope of 24 dB/octave. You may not even notice the drift if you only look at short EEG epochs (e.g., 5 s or less), and it's a good idea to look at longer time periods (e.g., 60 s or more) so that you can see the slow drifts in your data. As

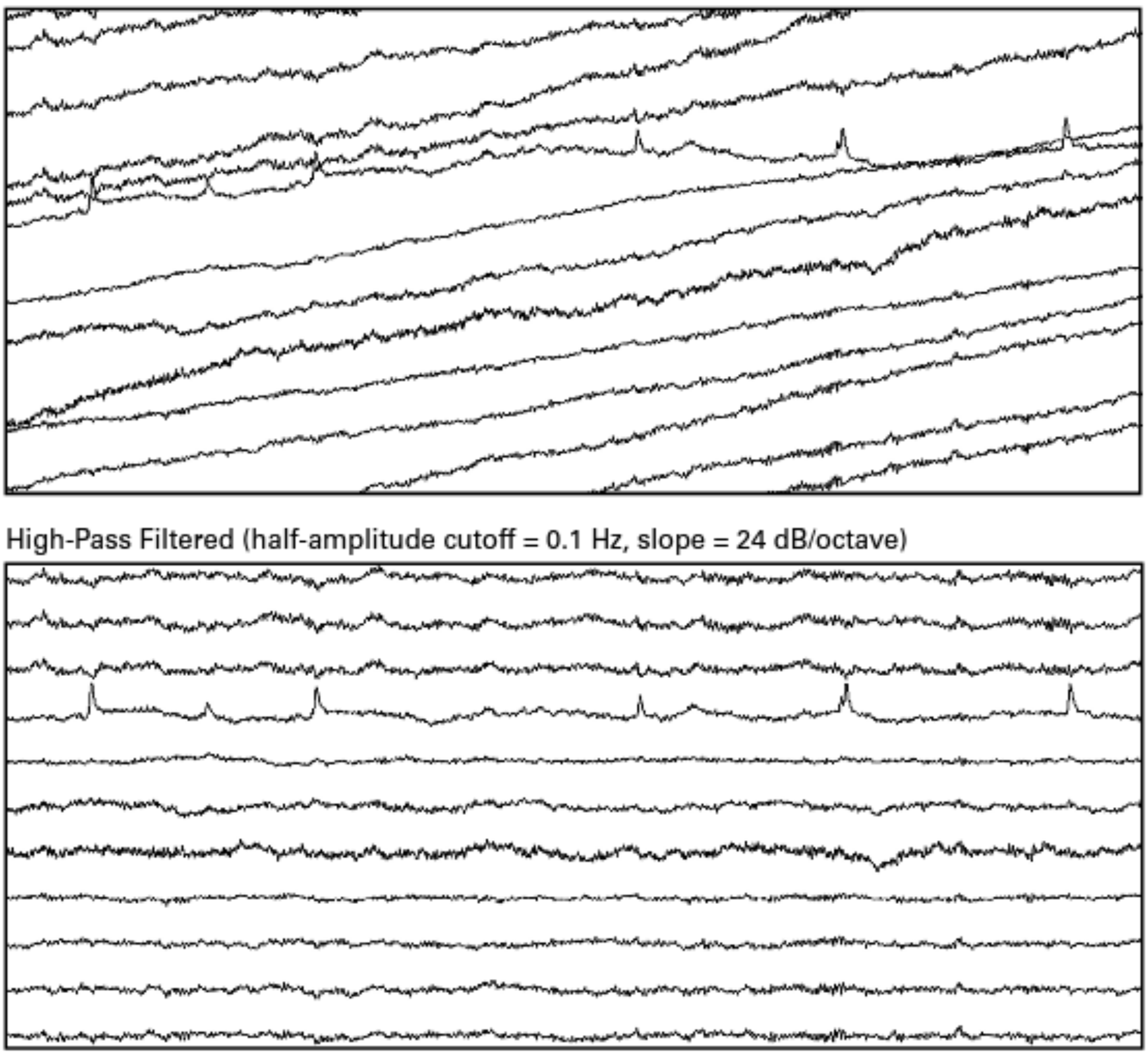

Figure 7.6
Example of a 30-s period of EEG without any filtering (top) and after the application of a high-pass filter with a half-amplitude cutoff of 0.1 Hz and a slope of 24 dB/octave (bottom). The filter eliminates the gradual upward drift in the data.

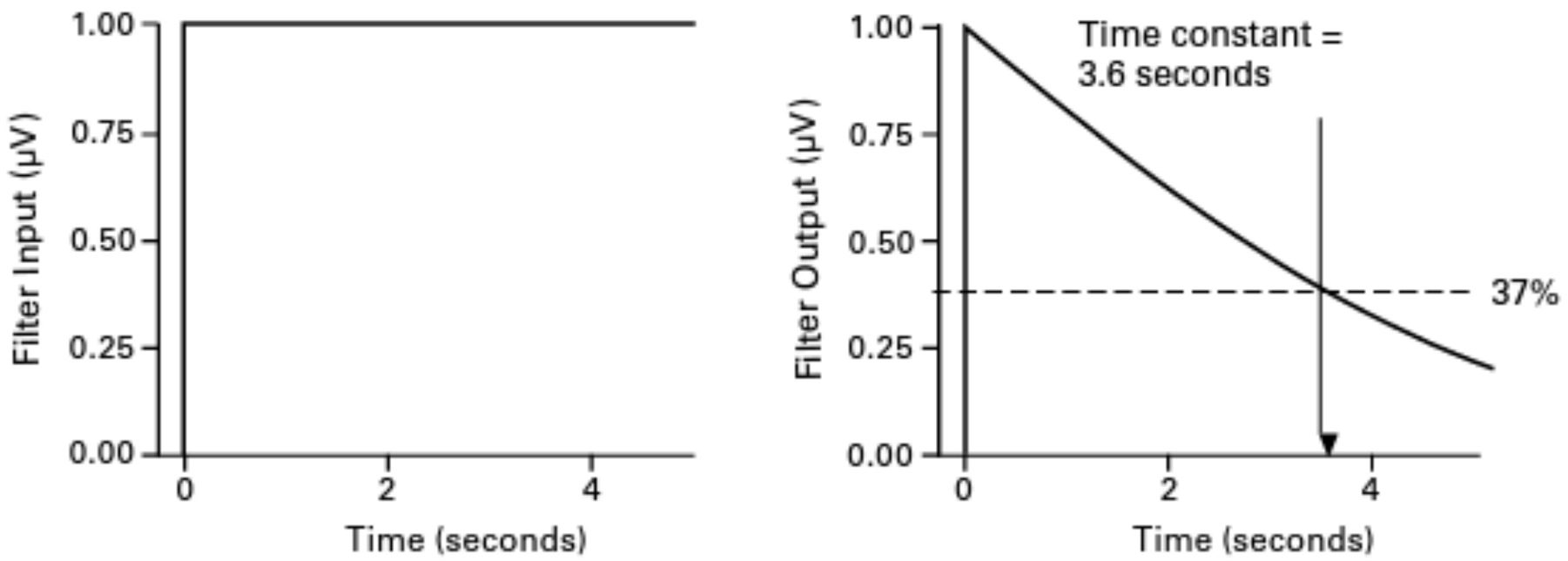

Figure 7.7
Example of the time constant of a high-pass filter. If the input to the filter is a constant voltage (left), the output of the filter will fall gradually toward zero according to an exponential function. The time constant of this particular filter is 3.6 s, which means that the output of the filter at a given time point will be 1/*e* (37%) of the value 3.6 s previously. As long as the input to the filter is a constant voltage, this percentage-based drop will be true over any 3.6-s period.

discussed in the section "When Should You Filter?" near the end of the chapter, high-pass filters are usually applied to the continuous EEG rather than to the epoched EEG or to averaged ERPs.

High-pass filters are sometimes described in terms of the *time constant* rather than the half-amplitude cutoff. As shown in figure 7.7, if the input to a high-pass filter is a constant voltage, the output of the filter will start at this voltage and then gradually fall toward zero. The filter's time constant is a measure of the rate at which it causes the voltage to fall. The decline in output voltage over time is exponential, which means that the voltage drops by a particular percentage between the beginning and end of a time period of a given length (e.g., it might drop by 50% between the beginning and end of an 8-s period, and then by 50% of the remaining voltage by the end of the next 8-s period). Because the drop in voltage over a period of time is always a percentage of the voltage at the beginning of that period, the voltage never quite reaches zero. Consequently, the time constant is expressed as the time required for the filter's output to reach a particular proportion (1/*e*, or 37%) of the starting value. It's called a "constant" because, for any starting point, the voltage will drop to 1/*e* of the starting value in that amount of time.

As the half-amplitude cutoff becomes higher, the time constant becomes shorter. If you know the half-power cutoff frequency of a high-pass filter (f_c, the frequency at which the filter's output is reduced by 3 dB), the time constant can be computed as $1/(2\pi f_c)$.

Basics of Filtering in the Time Domain

In conventional ERP analyses, frequency per se is not an important variable for analysis. Instead, ERP studies usually focus on the time course of the brain activity. However, ERP researchers almost always talk about the frequency-domain properties of filters rather than their time-domain properties. I find this a little odd. In this section, I will therefore explain how filters operate as

a time-based process, without any frequency-domain representations. I will focus on a particularly simple type of time-domain filter called a *running average* filter, which won't require any math beyond simple averaging. Online chapter 12 provides a more generalized description of how filters operate in the time domain, which I encourage you to read if you want to go beyond the filtering recommendations provided at the end of this chapter (or if you someday hope to become an ERP guru).

Low-Pass Filtering with a Running Average Filter

Let's start by looking at the ERP waveform in figure 7.8A. It has a lot of high-frequency noise (small rapid changes in voltage that make the waveform look a bit fuzzy). A simple way to

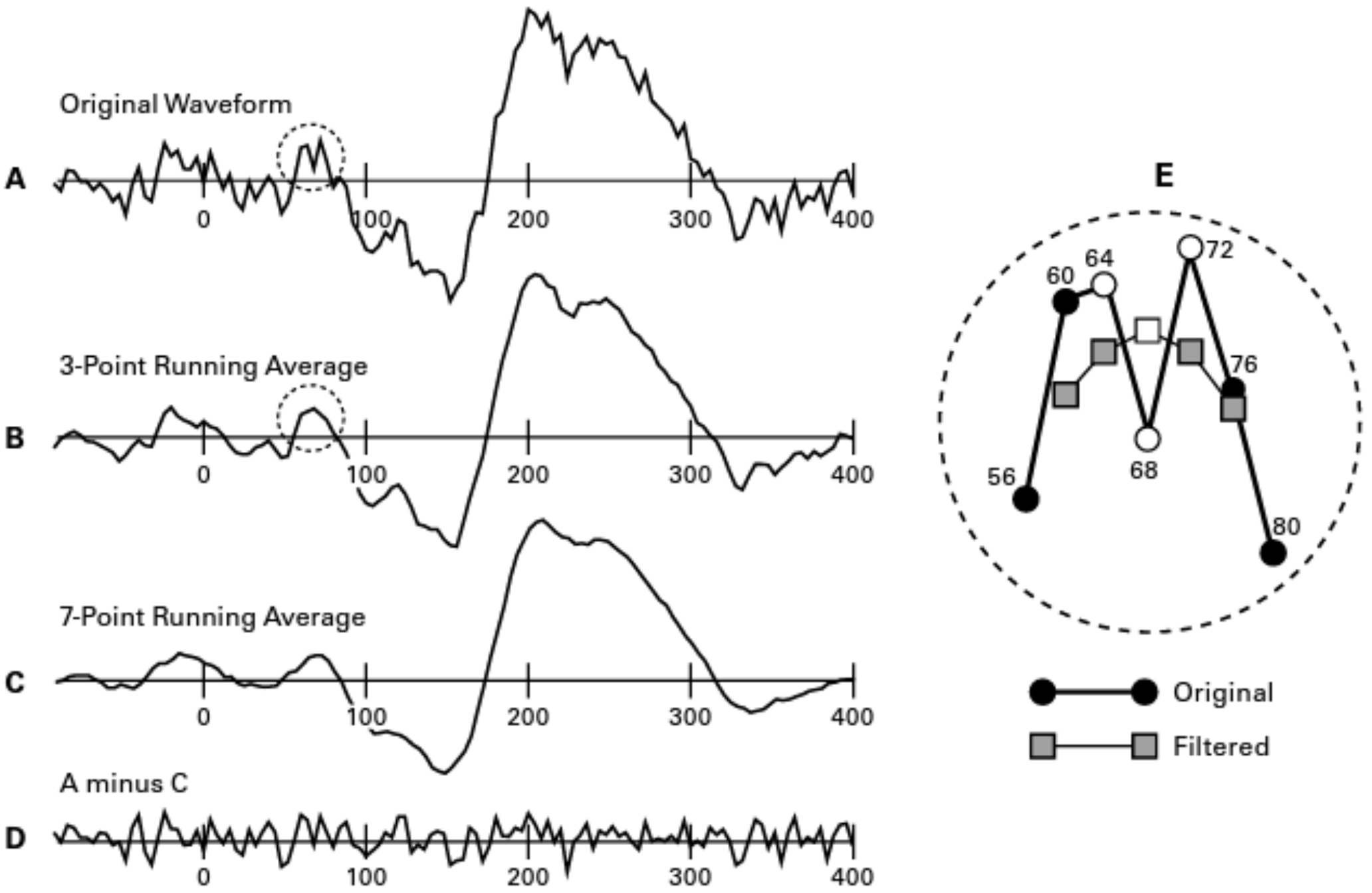

Figure 7.8
Example of filtering an ERP waveform with a running average filter, which works by averaging together the voltages surrounding each time point. (A) Unfiltered ERP waveform, contaminated by substantial high-frequency noise. (B) Result of filtering the waveform in panel A by averaging the voltage at each time point with the voltages at the immediately adjacent time points (a three-point running average filter). (C) Result of filtering the waveform in panel A by averaging the voltage at each time point with the voltages at the three time points on either side (a seven-point running average filter). (D) High-pass–filtered waveform, constructed by subtracting the filtered waveform in panel C from the unfiltered waveform in panel A. (E) Close-up of the original and filtered data between 56 and 80 ms. The original values are indicated by circles, and the filtered values are indicated by squares. Each number represents the time point of the value. The three white-filled circles were averaged together to create the white-filled square.

reduce these rapid changes would be to replace each point in the waveform with the average of the voltage at that point and the voltages at the time points immediately before and immediately after. For example, if we have one sample every 4 ms, the filtered value at 68 ms would be equal to the average of the unfiltered voltages at 64, 68, and 72 ms. Similarly, the filtered value at 72 ms would be equal to the average of the unfiltered voltages at 68, 72, and 76 ms. This would be called a *three-point running average filter* because the filtered waveform is computed by taking a running average of every three points in the original waveform. The result of applying this simple procedure is shown in figure 7.8B. The filtered waveform looks smoother than the original waveform in figure 7.8A, but it is otherwise the same basic waveform.

Just to make sure this is clear, figure 7.8E shows a blown-up view of seven samples from the original waveform (the circled part of the waveform in figure 7.8A, from 56 to 80 ms) and five of the corresponding samples from the filtered waveform (the circled part of the waveform in figure 7.8B, from 60 to 76 ms). The filtered value at 68 ms (the white square) is simply the average of the points at 64, 68, and 72 ms in the original waveform (the three white circles). Similarly, the filtered value at 64 ms is the average of the original values at 60, 64, and 68 ms.

I am showing you seven values from the original waveform, but I have shown you only five points of the filtered values, leaving out the first and last points in this time period. This is because filters run into a problem at the beginning and end of the waveform. To compute the filtered value at 56 ms, you would need to know the original value at 52, 56, and 60 ms. Similarly, to compute the filtered value at 80 ms, you would need to know the original value at 84 ms (and average this with the original values at 76 and 80 ms). Because I haven't shown you the original values at 52 and 84 ms, it's hard to show you the filtered values at 56 and 80 ms. Of course, if we look at the entire original waveform, we can find the values at 52 and 84 ms and compute the filtered values at 56 and 80 ms. However, this problem will arise at the very beginning and very end of the overall waveform (at –100 and +400 ms), and the filtered waveform becomes undefined at these time points. I will return to this later, because the same underlying problem arises with frequency-domain filters as well.

Figure 7.8C shows the same approach to filtering, but using a seven-point running average rather than a three-point running average. For example, the filtered value at 68 ms was computed as the average of the original values at 56, 60, 64, 68, 72, 76, and 80 ms. This waveform is even smoother than the waveform created with the three-point running average. The longer the running average, the more heavily you are filtering the data. As the next section describes in more detail, increasing the number of points in a running average is mathematically equivalent to decreasing the half-amplitude cutoff of a frequency-domain filter.

Relationship to Frequency-Domain Filtering

You are probably wondering how this type of filtering is related to frequency-domain filtering. As will be described in detail in online chapter 12, any time-domain filter has an equivalent

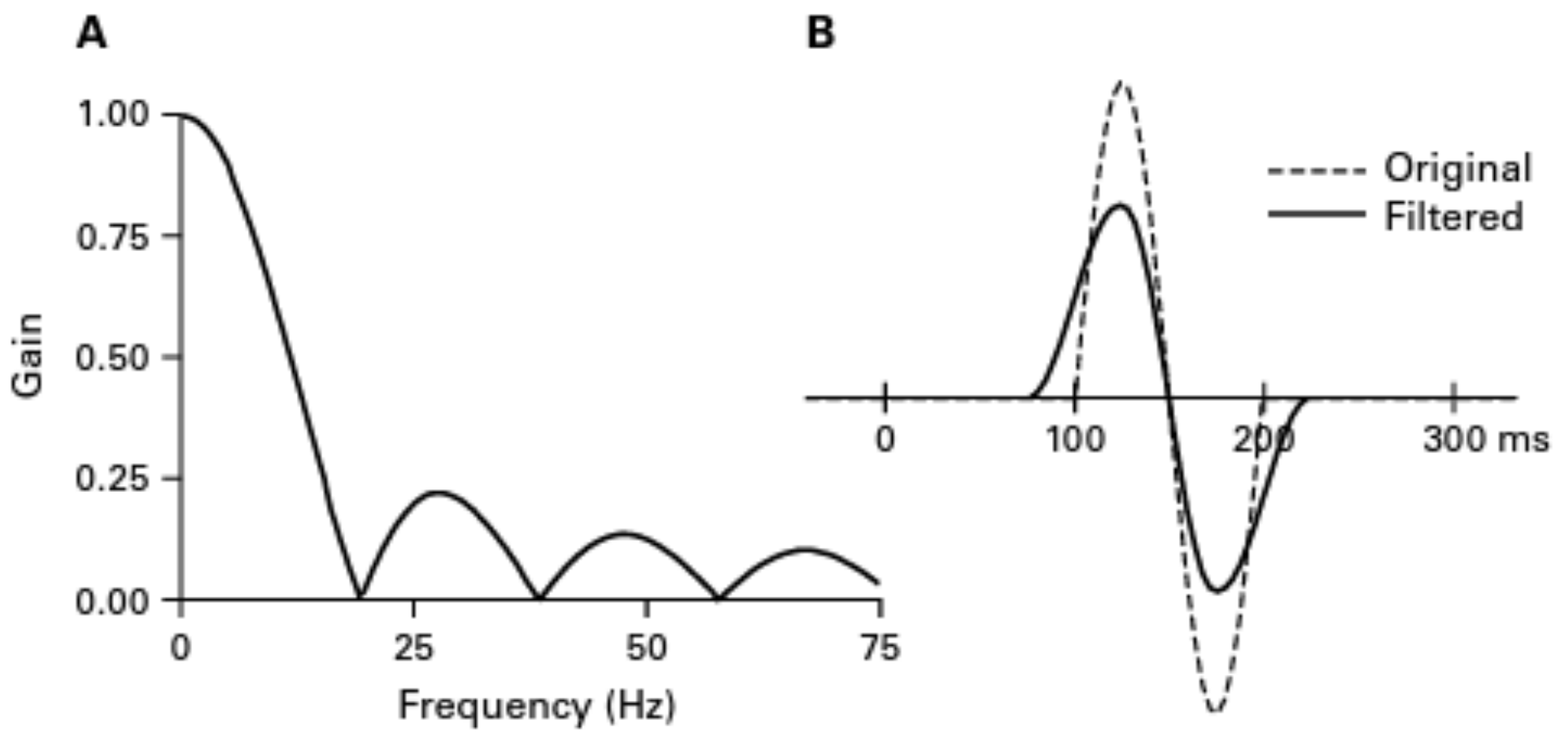

Figure 7.9
(A) Frequency response function of a 13-point running average filter (assuming a sampling rate of 250 Hz). (B) Artificial waveform consisting of one cycle of a 10-Hz sine wave beginning at 100 ms (dashed line), and the result of filtering this waveform with a 13-point running average filter (solid line).

frequency-domain filter, and vice versa. Figure 7.9A shows the frequency response function of a 13-point running average filter (assuming a sampling rate of 250 Hz). It passes low frequencies perfectly, and then the gain drops off as the frequency increases, falling to zero at 20.83 Hz. The gain then rises up a little and falls back to zero again at 41.67 Hz. This up-and-down pattern continues infinitely, getting smaller and smaller as the frequency increases. If you filtered an ERP waveform using the frequency-domain approach to filtering (e.g., converting the ERP waveform to the frequency domain, multiplying it by this frequency response function, and then converting it back into the time domain), the result would be exactly the same as applying a 13-point running average filter.

The frequency response function of a running average filter is a little strange because it goes up and down multiple times. However, it is possible to make a slight change to the running average filter and get a much nicer, monotonically decreasing frequency response function. This change also addresses a time-domain oddity of the running average filter. Specifically, when we compute the filtered value at a given time point with a 13-point running average filter, we give equal weight to all six time points on either side of the current time point. It might make more sense to give a higher weight to the time points that are closer to the current time point. With a three-point running average filter, for example, we could give the current point a weight of 0.5 and each of the surrounding points a weight of 0.25. The filtered value at 68 ms, for example, would be 0.25 times the voltage at 64 ms plus 0.5 times the voltage at 68 ms plus 0.25 times the voltage at 72 ms. By choosing an appropriate set of weights, you can create a "weighted" running average filter that has a monotonically decreasing frequency response function. In fact, by choosing appropriate weights, you can create any frequency response function you desire. The details of this are described in online chapter 12.

Filtering and Temporal Smearing

When we filter with a running average filter, it's clear that we are losing some temporal resolution. That is, because the filtered voltage at a given time point is computed by averaging the surrounding time points, the filtered value represents the average activity over a range of time points rather than representing activity at a single instant. This is illustrated in figure 7.9B, which shows what happens when we apply a 13-point running average filter to an artificial waveform (a single cycle of a 10-Hz sine wave that begins at 100 ms). The filtered waveform has a lower overall amplitude than the original waveform, which makes sense given that the gain at 10 Hz is well below 1.0 in the frequency response function shown in figure 7.9A. However, the most notable thing about the filtered waveform is that it has an earlier onset time than the original waveform (and a later offset time as well). This makes sense given how a running average is computed. For example, the filtered value at 92 ms is the average of the voltages between 68 and 116 ms (this is 13 points given that we have one point every 4 ms). Because the original waveform starts rising above zero in this time range, the average of these voltages is above zero. This causes the filtered value to be above zero at 92 ms, even though the original waveform does not deviate from zero until after 100 ms.

You might think that this would be a reason not to filter with a running average filter and instead use a frequency-domain filter. However, time-domain and frequency-domain filters are mathematically equivalent, and this same kind of "smearing" of temporal information is produced when ERPs are filtered in the frequency domain. I will return to this later in the chapter.

High-Pass Filtering with a Running Average Filter

You now know how to perform a simple low-pass filtering operation. You can perform high-pass filtering in the same way by adding a simple trick. This trick takes advantage of the fact that the overall ERP waveform is equivalent to the sum of the low frequencies plus the high frequencies, whereas the low-pass–filtered waveform contains just the low frequencies. If we take the original waveform (the high frequencies plus low frequencies) and subtract the low-pass–filtered waveform (just the low frequencies), the result is a waveform that contains only the high frequencies. In other words, we can create a high-pass–filtered waveform by subtracting a low-pass–filtered waveform from the original data. This is shown in figure 7.8D, which is the original waveform minus the waveform that was filtered with the seven-point running average filter. If you look closely, each little upward or downward blip in the high-pass–filtered waveform corresponds to a little upward or downward blip in the original waveform. This particular high-pass filter, which used a seven-point running average, has such a high half-amplitude cutoff that you would never want to use it. However, you could achieve a more useful half-amplitude cutoff by subtracting a waveform that was filtered with, for example, a 101-point running average.

The unfiltered waveform shown in figure 7.8A contains only 125 points (500 ms at a sampling rate of 250 Hz). If you applied a 101-point running average filter, you would be unable to compute the filtered value for the first and last 50 points in the waveform (because you need 50 points on either side of a given point to compute the average of 101 points). That would leave

us with only the middle 25 points in the filtered waveform, which is obviously a problem. Consequently, a filter with a lot of points should ordinarily be applied to the continuous EEG (i.e., prior to epoching). Losing the first 50 and last 50 points of a 5-min period of continuous EEG is not much of a problem. In fact, you could just start the recording a little bit before the stimuli start and end it a little after the stimuli end, and then you wouldn't lose any important data by filtering.

Distortions Produced by Filtering

Although filters are extremely useful, they are really a form of systematic distortion. The more heavily you filter the data, the more you are distorting your data. The distortions caused by filtering can be summarized by a key principle that you should commit to memory and recall every time the topic of filtering comes up.

Fundamental principle of filtering Precision in the time domain is inversely related to precision in the frequency domain.

In other words, the more tightly you constrain the frequencies in an ERP waveform (i.e., by filtering out everything but a narrow band of frequencies or by using steep roll-offs), the more the ERP waveform will become spread out in time. Conversely, the more temporal precision you have, the broader the range of frequencies will be in your data. Given that temporal resolution is one of the main virtues of the ERP technique, this principle makes it clear that we give up a lot if we try to be highly precise about the frequency information.

To make this principle more concrete, figure 7.10 shows three types of temporal spreading that can be produced by filtering a realistic ERP waveform. These are all fairly extreme examples,

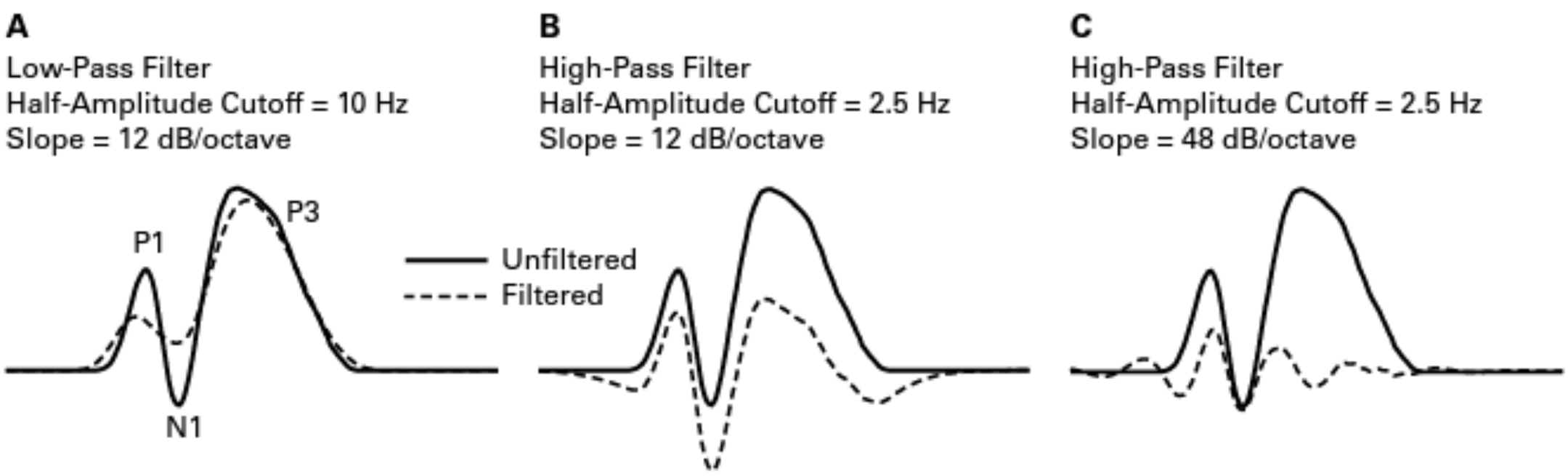

Figure 7.10
Examples of distortions caused by filtering. (A) Effects of low-pass filtering on the onset and offset times of an ERP waveform. (B) Effects of a high-pass filter with a relatively gentle roll-off. Note the artificial negative peaks at the beginning and end of the filtered waveform. (C) Effects of a high-pass filter with a relatively steep roll-off. Note the artificial oscillations in the filtered waveform.

and you should not see this kind of distortion in your own data if you follow the recommendations at the end of the chapter. Figure 7.10A shows how low-pass filtering an ERP waveform (half-amplitude cutoff = 10 Hz) causes the filtered waveform to start earlier and end later than the unfiltered waveform. This is similar to the smearing that we saw with a 13-point running average filter in figure 7.9B. Whether implemented in the time domain or in the frequency domain, low-pass filters always have this smearing effect, which may affect the onset and offset times of the ERP components and experimental effects. For example, if you see an experimental effect beginning at 120 ms in a heavily low-pass–filtered waveform, the effect may have actually started quite a bit later (e.g., 150 ms).

Figure 7.10B shows the effect of a high-pass filter with a half-amplitude cutoff at 2.5 Hz and a relatively gradual slope of 12 dB/octave. This panel shows that the spreading of voltage produced by high-pass filtering is inverted in polarity. The spread is inverted because a high-pass filter is equivalent to subtracting a low-pass–filtered waveform from the original data (as described earlier in the chapter). The smearing produced by the low-pass filter is inverted by the subtraction. In the example shown here, the positive P1 and P3 peaks at the beginning and end of the unfiltered waveform lead to artifactual negative peaks at the beginning and end of the filtered waveform. The N1 peak in the middle of the waveform also induces artificial positive activity at surrounding time points, but this just blends into the P1 and P3 peaks and is therefore difficult to see in this particular example. However, if two conditions differed only in N1 amplitude, this type of filtering could lead to artificial positive effects during the P1 and P3 peaks.

Figure 7.10C shows the distortion caused by a high-pass filter with the same half-amplitude cutoff (2.5 Hz) but a sharper slope (48 dB/octave). Instead of individual artifactual peaks at the beginning and end of the filtered waveform, we now have artifactual oscillations that extend even further in time. This sort of artifactual oscillation could also cause an experimental effect to appear to oscillate. As you can imagine, the use of this sort of filter might cause someone to completely misinterpret the results of an ERP experiment. Box 7.2 describes an influential study that reported oscillations that may have been a result of this kind of filter artifact.

Figure 7.11A shows how increasing the high-pass cutoff can attenuate the P3 wave and create an artificial peak at the beginning of the waveform (from the study of Kappenman & Luck, 2010). We found that a high-pass filter with a 0.1-Hz cutoff attenuated the P3 wave only slightly, and it did not produce any discernible artifacts. A filter with a 0.5-Hz cutoff clearly attenuated the amplitude of the P3 wave, and it also produced an artifactual negative deflection from approximately –50 to +100 ms. A filter with a 1.0-Hz cutoff reduced the amplitude of the P3 wave even more, and it also increased the amplitude of the artifactual negative deflection at the beginning of the waveform. I hope this makes it clear that high-pass filters can be dangerous when the half-amplitude cutoff is above approximately 0.1 Hz.

If you have already filtered some data, you may be wondering if your filters produced significant artifacts in your waveforms. Similarly, you may be wondering how you can tell if a given filter produces artificial peaks or oscillations. The best way to determine how a filter distorts the data is to apply the filter to a known, artificial signal. Practical directions for accomplishing this

Box 7.2
An Example from the Literature

Luu and Tucker (2001) published an influential study of the ERN in which they used relatively severe filtering in an attempt to demonstrate that the ERN is a modulation of an ongoing oscillation rather than being a discrete, transient neural response. The original waveforms for correct and error trials (time-locked to the response) are shown on the left side of the figure in this box, and the filtered data (band-pass of 4–12 Hz) are shown on the right side. When I first saw these waveforms, I thought to myself, "the filtered data look a lot like the examples of artificial oscillations that I show in my book" (see, e.g., figure 7.10C). In other words, the oscillations in the filtered data may have been an artifact of the filters rather than a true feature of the underlying brain activity.

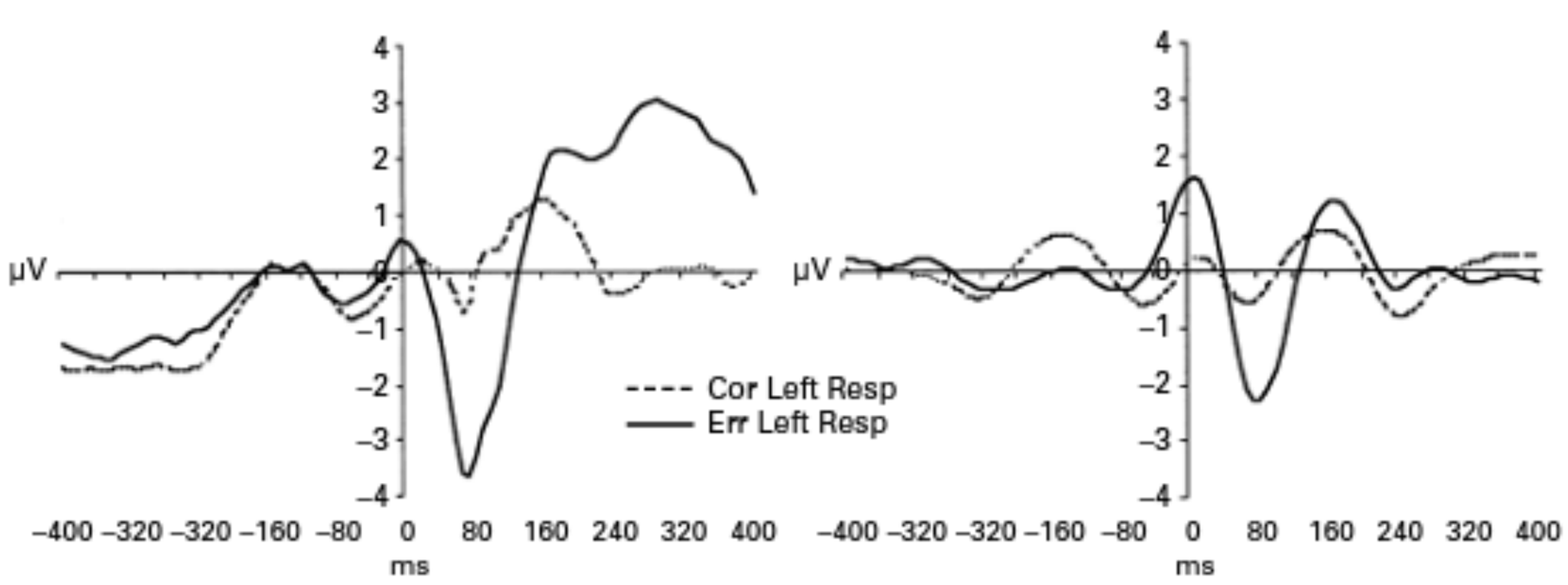

Figure 1 from Luu and Tucker (2001). Reprinted by permission. Copyright 2001 by Elsevier Science Ireland Ltd.

I planned to run some simulations to demonstrate that the pattern of results observed by Luu and Tucker (2001) could reflect a filter artifact, but then I discovered that Nick Yeung and his colleagues had already published a paper making this point (Yeung, Bogacz, Holroyd, Nieuwenhuis, & Cohen, 2007). That is, they conducted simulations demonstrating that the apparent oscillations in the filtered data of Luu and Tucker (2001) could be explained either by a true oscillation or by a filter artifact. This doesn't mean that the conclusions of Luu and Tucker (2001) were incorrect; it just means that their filtering procedure didn't provide any real evidence about whether an oscillation was present. If you filter EEG or ERP data with a narrow band-pass, you will almost always see oscillations, whether or not the underlying brain signal is oscillating.

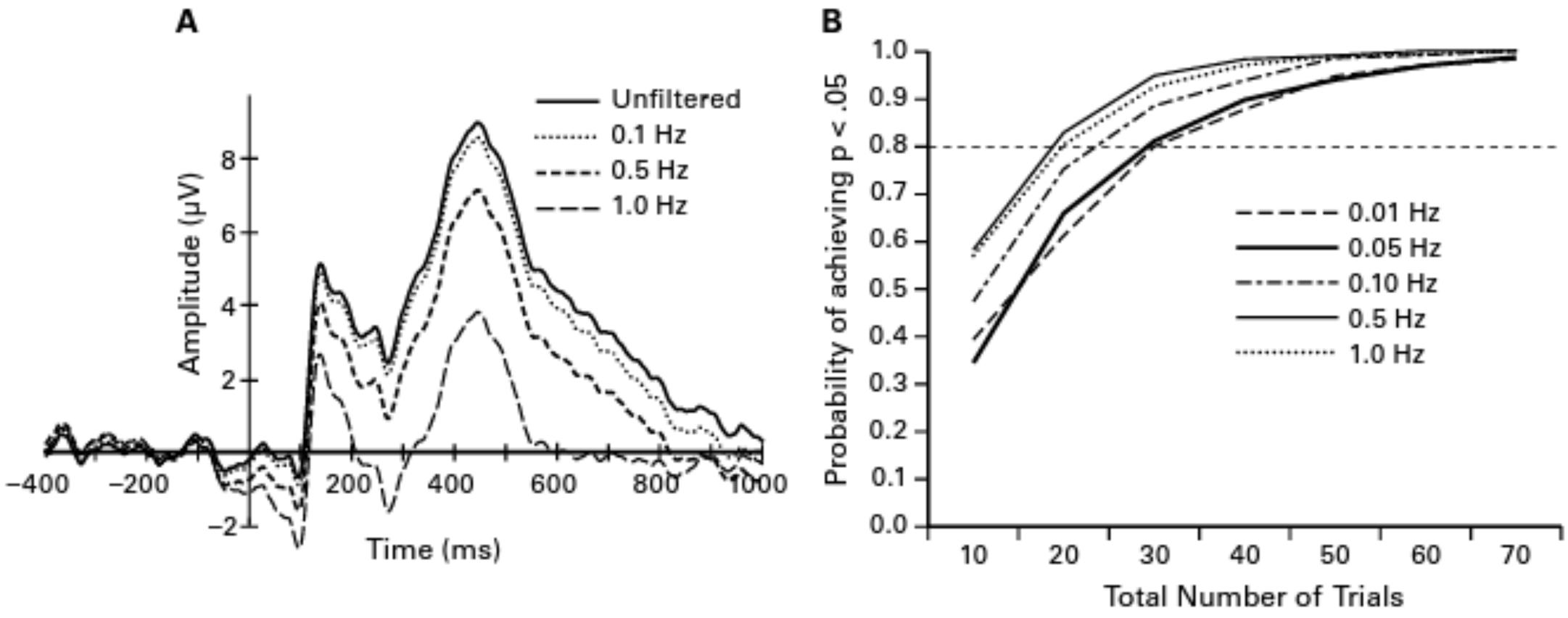

Figure 7.11
(A) Effects of different high-pass filters on the ERP waveform elicited by a visual oddball stimulus. The cutoffs indicate the half-amplitude value of a noncausal Butterworth filter with a slope of 24 dB/octave. (B) Effect of number of trials and high-pass cutoff frequency on the probability of obtaining a significant difference in amplitude between the rare targets and the frequent standards (from high-impedance recordings in a cool and dry recording environment). Adapted from the study of Kappenman and Luck (2010).

are provided in box 7.3. I strongly recommend that you give this a try. You will either learn that everything is okay (which will make you feel better) or that you need to reanalyze your data with different filters (which will allow you to avoid incorrect conclusions).

Do You Really Want to Use a High-Pass Filter?

Under most realistic conditions, high-pass filters are more likely than low-pass filters to produce incorrect conclusions. This is because the spreading of the onset and offset produced by low-pass filters is relatively modest (as long as the cutoff isn't too low) and typically influences all groups and conditions equally. In other words, the absolute onset time of an effect may appear to be slightly early, but this shift should be similar for all the waveforms in a given study. High-pass filters, in contrast, are more likely to be used with cutoffs that produce noticeable distortion, and they can create artificial peaks in the waveforms.

You may therefore wonder if you should avoid using high-pass filters. If you use a high-pass filter with a half-amplitude cutoff of 0.1 Hz or less, it's unlikely that the filter will cause any significant distortion, but it may substantially increase your statistical power, especially if you are looking at relatively late components such as P3 and N400. This is illustrated in figure 7.11B, which shows the results from a set of Monte Carlo simulations that Emily Kappenman ran with the P3 data shown in figure 7.10A (Kappenman & Luck, 2010). Statistical power (the probability of obtaining statistical significance when there truly is an effect) increased as the number of trials per subject increased, which was no surprise. The key finding was that statistical power

Box 7.3
Are Your Filters Distorting Your Data?

If you've already filtered the data for previous experiments, you may be wondering whether you significantly distorted your data. Similarly, if your advisor insists that you filter your data in a way that is inconsistent with my recommendations, you might be worried that you will distort your data. Fortunately, there is an easy way to determine how much your filters are distorting your data (assuming that the filtering in question is being conducted offline by a reasonably flexible software package). Specifically, you can create artificial waveforms that represent what you think your data might look like without any noise, and you can see how your filters distort these artificial waveforms.

To do this, you can create the artificial waveforms in a spreadsheet program (e.g., Microsoft Excel), save them as text files, import them into your analysis system, and apply your filters. To make this a little easier for you to do, I've provided an Excel spreadsheet, the corresponding text file, and an ERPLAB data file (available online at http://mitpress.mit.edu/luck2e). If you are using ERPLAB, you can just load the ERPLAB file and start filtering. If you are using a different analysis system, you can probably import the text file directly into your system. It will probably take you less than an hour of work to see how your filters distort the data. I strongly recommend that you take the time to do this. At a minimum, you may be relieved to see that your filters are producing negligible distortion. But you might find that your filters are producing severe distortion, and this exercise may prevent you from publishing an incorrect conclusion or might help you convince your advisor to follow my filtering recommendations.

for a given number of trials was substantially improved by applying a high-pass filter with a half-amplitude cutoff of 0.1 Hz compared to a lower cutoff frequency (0.01 Hz) or to unfiltered data (not shown here). Power was even better for higher cutoff frequencies (0.5 or 1.0 Hz). However, these higher cutoffs led to substantial distortion of the waveforms (as shown in figure 7.10A). Thus, 0.1 Hz appears to provide the best balance between statistical power and waveform distortion for the P3 wave.

Two additional findings from this study should be noted. First, the effects shown in figure 7.11B were obtained with data from high-impedance recordings. As discussed in chapter 5, low-frequency noise is typically much more of a problem when the electrodes impedances are high than when they are low. Emily found that filtering had much less of an effect on power in low-impedance recordings, presumably because there was less low-frequency noise to be filtered. Second, high-pass filtering had very little effect on statistical power when N1 amplitude was analyzed instead of P3 amplitude. As will be discussed in chapter 8, baseline correction serves as a form of high-pass filtering, but it becomes progressively less effective at filtering low frequencies later in the epoch (see especially figure 8.2D). The N1 wave occurs very soon after the baseline period, so additional filtering of low frequencies does not make much difference. The P3 is farther away from the baseline period, and it is therefore more affected by slow voltage drifts. Thus, high-pass filters are especially useful when you are measuring later components such as P3, N400, and LPP.

Recommendations for Filtering

Now that I've discussed why filters are used, how they work, and how they can distort your data, I will provide some concrete recommendations about filtering. These recommendations may not be appropriate for every experiment, but they will work well for the vast majority of ERP experiments in cognitive and affective neuroscience. If you want to filter your data more heavily than I recommend here, you need to understand the details of how filters work (e.g., by reading online chapter 12 and making sure that you understand all the math), and you should try filtering artificial waveforms to see the distortions produced by your filters (see box 7.3).

Before you start filtering your data, you should remind yourself of Hansen's axiom: *There is no substitute for clean data* (see chapter 5). Some minor filtering is necessary when the data are first being collected, and a modest amount of additional offline filtering is usually a good idea. However, filters cannot help very much if your data are noisy because of variability across subjects, variability across trials, a small number of trials in your averages, and so forth. Filters may make the data *look* better under these conditions, but this may be an illusion that could lead you to draw incorrect conclusions.

Online Filter Recommendations

In general, you should do only minimal filtering online (i.e., hardware filtering during data acquisition). It's always possible to filter the data more offline, but you can't "unfilter" data that have already been filtered. Moreover, the filters that can be applied in software offline are superior to online filters (e.g., online filters will produce a latency shift, but most offline filters will not).

No matter what kind of EEG recording system you are using, you will definitely need to use a low-pass filter to prevent aliasing during data acquisition. The half-amplitude cutoff should be between one-third and one-fifth of the sampling rate so that the gain of the frequency response function is near zero at the Nyquist frequency (one-half the sampling rate). Many systems will automatically select an appropriate anti-aliasing filter when you select the sampling rate.

In many situations, an anti-aliasing filter is the only online filter I would recommend using. Some systems allow you to apply additional filters to the EEG that you are viewing during data acquisition, without actually applying these filters to the data that are being saved to disk. This is a really great option because it's easier for you to monitor the data for artifacts and bad connections if you are viewing filtered data, but it's always better to do the final filtering in software offline. If your system has this ability, I would recommend viewing the EEG with a band-pass of 0.1–30 Hz (i.e., a high-pass filter with a half-amplitude cutoff of 0.1 Hz and a low-pass filter with a half-amplitude cutoff of 30 Hz). You should also look at the unfiltered data occasionally to see if you have large skin potentials or high levels of line noise or EMG activity.

If your data acquisition system has fewer than 20 bits of resolution, you should apply a hardware high-pass filter prior to digitization (see chapter 5 for the rationale). Otherwise, large voltage offsets (due to skin potentials, movement artifacts, etc.) may cause your system to

saturate. If you are in this situation, you should use a half-amplitude cutoff somewhere between 0.01 and 0.1 Hz. If your subjects are prone to skin potentials and movement artifacts (e.g., children, neurological patients), I would recommend 0.1 Hz to avoid excessive data loss. With highly cooperative subjects (e.g., healthy young adults), I would recommend 0.01 Hz (with additional high-pass filtering offline).

If your data acquisition system does not allow you to view filtered data while saving the unfiltered data, you may need to apply a notch filter at the line frequency (50 or 60 Hz, depending on where you live). Notch filters are not ideal, but if you have high levels of line noise, you may not be able to monitor the EEG adequately without one. Of course, it's much better to minimize sources of electrical noise in the environment so that the EEG is not contaminated by line noise (as described in online chapter 16). However, this is not always possible (e.g., if you are recording from the bedside in a hospital).

Offline Filter Recommendations

For typical experiments on cognitive and affective processes (or perceptual processes that begin after approximately 50 ms poststimulus), I recommend adding offline filters, if necessary, to achieve a final band-pass of approximately 0.1–30 Hz. That is, you should end up with a half-amplitude high-pass cutoff of 0.1 Hz to attenuate skin potentials and other slow voltage changes and a half-amplitude low-pass cutoff of 30 Hz to attenuate line noise and EMG noise. You don't need to use these exact values; anything between 0.05 and 0.2 Hz for the low end and between 20 and 50 Hz for the high end will be fine. And I recommend a slope of between 12 and 24 dB per octave (a steeper slope is usually acceptable for the low-pass filter but not for the high-pass filter). As mentioned earlier, you should use noncausal rather than causal filters for your offline filtering, which will avoid latency shifts.

I would again like to stress that you should not use a narrower bandwidth than this unless you really know what you're doing. Otherwise you may end up drawing unjustified conclusions (as in the study described in box 7.2).

If you filter your data during data acquisition with a band-pass of approximately 0.1–30 Hz, there is no need to filter again offline. However, if you use a broader band-pass during data acquisition (e.g., 0.01–100 Hz), you can filter again offline to achieve a final band-pass of 0.1–30 Hz.

There are always exceptions to any rule, and here are some common exceptions to my basic filter recommendations:

- If you are looking at very slow or late components, like the CDA or LPP, the conventional advice would be to set your high-pass filter at a lower frequency (e.g., 0.01 Hz) to avoid attenuating the amplitude of the component you are trying to measure. However, the benefits of filtering out low-frequency noise will also be greatest for these late components, and a modest attenuation of the signal might be more than offset by a large attenuation of noise. Thus, you might want to try a lower cutoff, but I suspect you will ultimately find that 0.1 Hz (or perhaps 0.05 Hz) is the best compromise between attenuation of the signal and noise reduction.

- If you are quantifying component amplitudes by measuring the mean amplitude over a time window of at least 50 ms (e.g., quantifying P3 amplitude as the mean voltage from 300 to 500 ms), there is no need to apply a low-pass filter prior to the measurement. The use of a wide window will already attenuate the high-frequency noise (see chapter 9 for details).
- If you are measuring the onset latency of an ERP component (or some other highly noise-sensitive feature of the waveform), you may need an even lower cutoff for your low-pass filter (e.g., 10 Hz). However, you should not do this unless you first gain a fuller understanding of filtering (e.g., by reading online chapter 12 and by filtering some artificial waveforms).
- If you are interested in very fast sensory responses that occur within 50 ms of stimulus onset (e.g., the auditory brainstem responses), you will want to use higher cutoff frequencies for both the low-pass and high-pass filters. To choose the precise frequencies, you should read papers in your area to see what other people use.

When Should You Filter?

During ERP Boot Camps, I am almost always asked whether filters should be applied before or after epoching, before or after artifact rejection, before or after averaging, and so forth. The answer usually depends on the particulars of an experiment, but you can make the right choice if you understand a fundamental principle of data processing: *the order of operations does not matter for linear operations*.

This principle is fully explained in the appendix of this book (including the meaning of the term *linear operations*). The basic idea is that some sorts of mathematical procedures give you the same result no matter the order in which they are applied. For example, (A + B) + C is equal to A + (B + C). With more complicated operations, such as filtering, the order of operations does not matter if the operations are linear. For example, if X and Y are linear operations, you can apply operation X to an ERP waveform and then use the resulting waveform as the input to operation Y, and this will yield the same end result as applying operation Y to the ERP waveform and then using the resulting waveform as the input to operation X.

Many of the filters that are used with EEG and ERP data are *finite impulse response* filters, and these are linear. Others are *infinite impulse response* filters (e.g., Butterworth filters), and these are nonlinear. However, infinite impulse response filters are approximately linear when the band-pass is fairly broad and the slope is fairly gentle. Thus, if you follow my recommendations for filtering, you can apply filtering and other linear operations in any order and get exactly (or almost exactly) the same result. However, you will need to keep in mind one caveat, described in the next section.

A very simple example of this principle arises when you filter the data twice. In the example shown in figure 7.5, it doesn't matter if you apply filter A and then filter B or if you apply filter B and then filter A. Either way, the result is equivalent to a single filter with a frequency response function that is the product of the frequency response functions of A and B. Similarly, it doesn't matter if you first apply a low-pass filter and then a high-pass filter or if you first apply a high-pass filter and then a low-pass filter. And applying a low-pass filter and a high-pass filter sequentially is equivalent to applying a band-pass filter in a single step.

Averaging is also a linear operation. Consequently, it doesn't matter if you filter the EEG immediately before averaging or if you filter the ERP immediately after averaging. Re-referencing is also a linear operation (except in unusual cases), so you can filter either before or after you re-reference the data.

In contrast, artifact rejection is a nonlinear operation, and filtering the EEG and then performing artifact rejection will lead to different results than performing artifact rejection and then filtering. The simple rule is that you should filter prior to artifact rejection if the filtering makes it easier for you to detect artifacts, and you should filter after artifact rejection if the filtering makes it more difficult to detect artifacts. For example, if you have a large amount of 60-Hz noise in the data, this may make it difficult to detect blinks by means of an absolute voltage threshold or by means of the moving window peak-to-peak amplitude approach (see chapter 6). In this case, you might want to apply a low-pass filter with a half-amplitude cutoff of 30 Hz prior to artifact rejection. If your data are reasonably clean, you will get the same results (or nearly the same results) if you filter before or after artifact rejection.

Edge Artifacts

There is one important caveat about the order of operations, which is that filters may work improperly when applied to short segments of data (e.g., epoched EEG segments or averaged ERP waveforms). In particular, filters may produce *edge artifacts* at the very beginning and very end of the waveform. These artifacts occur because the filtered value at a given time point is computed by using the surrounding time points; some of these time points may not exist near the beginning and end of the waveform, which may cause the filter to work incorrectly. We already discussed this issue in the context of running average filters, but the same principle applies to any filter (whether applied in the time domain or in the frequency domain).

This problem is especially severe when the cutoff frequency is low (for both low- and high-pass filters) and when the roll-off is steep, because these factors require the use of more time points to compute the filtered data. A filter with a cutoff frequency of 30 Hz and a slope of 12 dB/octave does not require many time points surrounding the current time point, whereas a filter with a cutoff frequency of 0.1 Hz and a slope of 48 dB/octave requires many time points. In addition, it is difficult to accurately estimate low frequencies from short time periods. For example, you get only half a cycle of an 0.5-Hz sine wave in a 1000-ms averaged ERP waveform, making it difficult to accurately estimate and remove this frequency from the data. Thus, you should avoid filtering short epochs of data with low cutoff frequencies. It is usually fine to apply low-pass filters to epoched EEG or averaged ERPs, but high-pass filters should usually be applied to the continuous EEG. In general, I recommend applying a 0.1-Hz high-pass filter to the continuous EEG, and then applying a 30-Hz low-pass filter after averaging.

Low cutoff frequencies work reasonably well when applied to long periods of continuous EEG because the beginning and end of the EEG waveform represent a small proportion of the overall signal and because there will be many cycles of the very low frequencies. For example, if you filter a 5-min continuous segment of EEG (i.e., the EEG recorded during a single block

of trials), any edge artifacts will be a very small part of the overall EEG. To be extra careful, I recommend starting the EEG recording about 20 s prior to the beginning of the trial block and ending it about 20 s after the end of the trial block (making sure that you don't have a lot of movement or other artifacts during these periods. That way, any edge artifacts will occur during a period that does not have any event codes and will not contribute to your ERPs.

Infinite impulse response filters (e.g., Butterworth filters) do not need to use as many time points as finite impulse response filters. Consequently, it can be advantageous to use an infinite impulse response filter when you are filtering epoched EEG data or averaged ERP data. For this reason, ERPLAB Toolbox mainly implements Butterworth filters.