# 13
# Filters: Digital Filters

## 13.1 INTRODUCTION

With currently available fast processors and dedicated digital signal processing (DSP) hardware, most biomedical instruments perform at least some filter operations in the digital domain (Chapter 2). In principle, this makes filtering more flexible; a different frequency response can be obtained with a simple change of parameters instead of requiring an alteration of the hardware. At first glance, it would seem that filtering in a ***digital*** world would allow arbitrary attenuation of undesired frequencies in the frequency domain representation of a signal. Unfortunately, there are limitations to this approach, since such manipulations in the frequency domain can introduce serious oscillations in the filter's response as well as unwanted transients in the time domain (Appendix 11.1).

## 13.2 IIR AND FIR DIGITAL FILTERS

In our analysis of continuous time LTI systems, we used a ***rational function*** to describe the input/output relationship in the time domain, the frequency domain, and the $s$ (Laplace) domain (Chapters 8 and 9). In discrete time, we can use the same approach for the sampled function using the $z$-domain instead of the $s$-domain, where we use time-delayed values instead of derivatives to characterize the evolution of the system. For a system with input $x(n)$ and output $y(n)$ with $n = 0, 1, 2, \ldots,$

$$\sum_{k=0}^{M} a_k y(n-k) = \sum_{k=0}^{N} b_k x(n-k) \tag{13.1}$$

with $a_k$ and $b_k$ as the parameters that determine the filter's characteristic. The $z$-transform can then be used to find the transfer function:

$$Y(z)\sum_{k=0}^{M} a_k z^{-k} = X(z)\sum_{k=0}^{N} b_k z^{-k} \rightarrow H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{N} b_k z^{-k}}{\sum_{k=0}^{M} a_k z^{-k}} \tag{13.2}$$

In some texts, the numerator and denominator are divided by $a_0$ (the coefficient of $y(n)$); this results in the following expression:

$$H(z) = \frac{\sum_{k=0}^{N} b_k z^{-k}}{1 + \sum_{k=1}^{M} a_k z^{-k}} \tag{13.3}$$

If a filter output depends on the previous output (i.e., $a_k \neq 0$ for $k \geq 1$), the response to an impulse at $n = 0$ never completely disappears but continues to reverberate through the system indefinitely. Because the impulse response continues forever ($n \rightarrow \infty$), this type of algorithm represents a so-called infinite impulse response (***IIR***) filter. In the case where $a_k = 0$ for $k \geq 1$, the output only depends on a finite set of input terms. Thus, the impulse response of this filter is finite: a finite impulse response (***FIR***) filter.

If we factor the polynomials in the numerator and denominator of (13.2), the rational function can also be fully characterized by a constant gain factor ($K$) plus the zeros of the numerator ($z_k$) and the zeros of the denominator, the so-called poles ($p_k$):

$$H(z) = \frac{Y(z)}{X(z)} = K\frac{(z^{-1} - z_1)(z^{-1} - z_2)\ldots(z^{-1} - z_n)}{(z^{-1} - p_1)(z^{-1} - p_2)\ldots(z^{-1} - p_m)} \tag{13.4}$$

It is easy to see that $H(z)$ is undefined at the poles, meaning an output/input ratio that explodes toward infinity. For a system to be stable, it must not have any poles in the so-called region of convergence (ROC, Appendix 9.2). Since an IIR filter equation includes poles, it is potentially unstable. In contrast, the FIR filters have no poles and are always stable.

## 13.3 AR, MA, AND ARMA FILTERS

An alternative classification of digital filters is based on the type of algorithm that is associated with the filter:

1. Autoregressive (***AR***) filters have a dependence on previous output and therefore are characterized by an infinite impulse response. An

example of such a (potentially unstable, depending on the coefficients) filter is

$$y(n) = Ay(n-1) + By(n-2) \quad (A \text{ and } B \text{ are constants}) \tag{13.5}$$

2. Moving average (***MA***) filters only depend on the input and therefore have a finite impulse response. An example of a moving average filter is

$$y(n) = \frac{x(n) + x(n-1) + x(n-2)}{A} \quad (A \text{ is a constant}) \tag{13.6}$$

3. The combination of AR and MA is the ***ARMA*** filter that depends on both previous output and input. The ARMA filter has an infinite impulse response because previous output is involved. An example of such a filter type is

$$y(n) = Ay(n-1) + Bx(n) + Cx(n-1) \quad (A, C, \text{ and } B \text{ are constants}) \tag{13.7}$$

As can be seen here, the AR, MA, and ARMA classifications overlap with the IIR and FIR terminology.

## 13.4 FREQUENCY CHARACTERISTIC OF DIGITAL FILTERS

The steps to transform a digital filter representation from the discrete time domain to the $z$-domain were shown earlier (e.g., Equations (13.1) and (13.2)). The $z$-transform of the output/input ratio (the transfer function) is closely related to the system's frequency response. In a digital filter's transfer function such as Equation (13.2), the variable $z$ represents $e^{s\tau}$ (Chapter 9, Section 9.5.2), where $s$ is a complex variable with a real component $\sigma$ and imaginary component $j\omega$ (Chapter 9, Section 9.3). For the frequency response, we are interested in the imaginary, frequency-related part of the transfer function. Therefore, we can determine the frequency response of a digital filter by substituting $e^{j\omega\tau}$ for $z$ in its transfer function.

This procedure was followed to obtain the frequency response in the example illustrated in pr12_4.m. In the following, we analyze an example of the 3-point smoothing (MA, FIR) filter in Equation (13.6) with $A = 3$. The $z$-transform of the time domain equation is $Y(z) = \frac{X(z)(1+z^{-1}+z^{-2})}{3}$, generating a transfer function:

$$\frac{Y(z)}{X(z)} = H(z) = \frac{(1 + z^{-1} + z^{-2})}{3} \tag{13.8}$$

Now we multiply the numerator and denominator by $z$, substitute $e^{j\omega\tau}$ for $z$, and use Euler's relation for $\cos(\omega\tau)$:

$$H(z) = \frac{(z + 1 + z^{-1})}{3z} \rightarrow H(j\omega) = \frac{(e^{j\omega\tau} + e^{-j\omega\tau} + 1)}{3e^{j\omega\tau}} = \frac{e^{-j\omega\tau}}{3}[2\cos(\omega\tau) + 1] \tag{13.9}$$

Remember that $\tau$ can be considered as the sample interval. This means that $1/\tau$ is the sample rate, $1/(2\tau)$ is the Nyquist frequency for the filter in Hz, and $\pi/\tau$ is the Nyquist frequency in rad/s. From the complex function in Equation (13.9), we can construct the Bode plot for values of $\omega$ ranging between 0 and $\pi/\tau$ rad/s. Use the following commands to calculate the expression in Equation (13.9) and to plot the output/input amplitude ratio of the Bode plot in MATLAB:

```
tau=1;                                  % sample interval
w=0:0.01:pi/tau;                        % rad Freq up to Nyquist
amp_ratio=abs((exp(-j*w*tau)/3).*(1+2*cos(w*tau)));
loglog(w,amp_ratio)                     % plot the result in log scales
```

If you prefer evaluating the result on a linear scale, you can use plot(w,amp_ratio) instead of the loglog command; the result you obtain using the plot command is shown in Figure 13.1. From the plot that is generated by these commands, it easy to see that the 3-point smoothing function behaves as a low-pass filter. Although this FIR filter is stable, the amplitude ratio of the frequency characteristic is far from ideal because there is a large side lobe above ($2\pi/3 \approx 2.1$ rad/s).

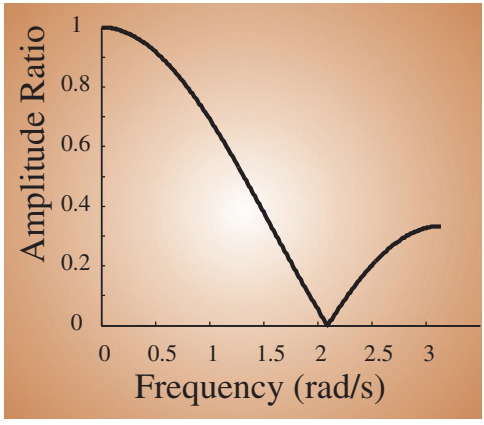



**Figure 13.1** Frequency characteristic of a 3-point smoothing filter. The amplitude ratio is plotted against the frequency.

## 13.5 MATLAB IMPLEMENTATION

The commands discussed in the following paragraphs are included in the MATLAB "Signal Processing" Toolbox. It is important to note that unlike most textbooks, MATLAB's vector indices start at 1 and not at 0!

The ***filter*** command requires the $a_k$ (A) and $b_k$ (B) coefficients for the digital filter operation on the input vector (e.g., X); the result is placed in another vector (e.g., Y). The following text shows the MATLAB help information for the filter command (type: help filter):

```
FILTER One-dimensional digital filter.
   Y = FILTER(B,A,X) filters the data in vector X with the
   filter described by vectors A and B to create the filtered
   data Y. The filter is a "Direct Form II Transposed"
   implementation of the standard difference equation:

   a(1)*y(n) = b(1)*x(n) + b(2)*x(n-1) +...+ b(nb+1)*x(n-nb)
         - a(2)*y(n-1) -...- a(na+1)*y(n-na)

   If a(1) is not equal to 1, FILTER normalizes the filter
   coefficients by a(1).

   FILTER always operates along the first non-singleton dimension,
   namely dimension 1 for column vectors and non-trivial matrices,
   and dimension 2 for row vectors.

   [Y,Zf] = FILTER(B,A,X,Zi) gives access to initial and final
   conditions, Zi and Zf, of the delays. Zi is a vector of length
   MAX(LENGTH(A),LENGTH(B))-1 or an array of such vectors, one
for
   each column of X.

   FILTER(B,A,X,[],DIM) or FILTER(B,A,X,Zi,DIM) operates along
the
   dimension DIM.

   See also FILTER2 and, in the Signal Processing Toolbox, FILTFILT.
```

Reprinted with permission of The MathWorks, Inc.

The vectors A and B contain the $a_k$ and $b_k$ coefficients that can be obtained directly or indirectly. For instance, if one wants to implement a filter,

$$y(n) - y(n-1) + 0.8y(n-2) = x(n) \tag{13.10}$$

The A and B coefficient vectors are

$$B = [1] \quad and \quad A = [1, -1, 0.8]$$

However, in most cases you do not know the A and B coefficients explicitly, and you have to instead start from a filter specification. For instance, we want to implement a band-pass filter that passes frequencies between 1 and 30 Hz. Suppose we are interested in implementing this by using a Butterworth filter (a special filter type; see also Section 13.6). We could do this the hard way by deriving the filter's transfer function and translating this into the discrete domain (Appendix 13.1). However, MATLAB allows one to determine the coefficients more easily using the butter command (type: help butter):

```
BUTTER Butterworth digital and analog filter design.
   [B,A] = BUTTER(N,Wn) designs an Nth order lowpass digital
   Butterworth filter and returns the filter coefficients in length
   N+1 vectors B (numerator) and A (denominator). The coefficients
   are listed in descending powers of z. The cutoff frequency
   Wn must be 0.0 < Wn < 1.0, with 1.0 corresponding to
   half the sample rate.

   If Wn is a two-element vector, Wn = [W1 W2], BUTTER returns an
   order 2N bandpass filter with passband W1 < W < W2.
   [B,A] = BUTTER(N,Wn,'high') designs a highpass filter.
   [B,A] = BUTTER(N,Wn,'stop') is a bandstop filter if Wn = [W1 W2].

   When used with three left-hand arguments, as in
   [Z,P,K] = BUTTER(...), the zeros and poles are returned in
   length N column vectors Z and P, and the gain in scalar K.

   When used with four left-hand arguments, as in
   [A,B,C,D] = BUTTER(...), state-space matrices are returned.

   BUTTER(N,Wn,'s'), BUTTER(N,Wn,'high','s') and
   BUTTER(N,Wn,'stop','s') design analog Butterworth filters. In this
   case, Wn can be bigger than 1.0.

   See also BUTTORD, BESSELF, CHEBY1, CHEBY2, ELLIP, FREQZ,
   FILTER.
```

Suppose we sampled our data at 400 Hz → the Nyquist frequency of the signal is 200 Hz. This means our bandwidth parameters should be 1/200 to 30/200.

The command [b,a] = butter(2, [1/200, 30/200]) produces the desired coefficients for a second-order filter. The coefficients can be used in the filter command to band pass a signal sampled at 400 Hz between 1 and 30 Hz.

Similarly, [b,a] = butter(6, [(60-5)/200,(60+5)/200],'stop') produces a set of coefficients that attenuate a 60-Hz noise component (a sixth-order band-reject filter between 55 and 65).

Another helpful feature in the Signal Processing Toolbox is the freqz command. This allows us to construct ***Bode plot*** from the filter characteristic in the z-domain. The plot is made on the basis of the coefficients A and B:

```
freqz(b,a,100,400)
```

In the preceding command, we pass parameters for the precision of the calculation (in this case, 100 points) and the sample frequency (400 in our example). The command impz shows the associated ***impulse response*** (e.g., impz(b,a,100) shows the impulse response of the first 100 points).

The file hum.mat (available on the CD; to load, type load hum) contains an epoch of EEG sampled at 256 Hz with a large 60-Hz component (after loading hum.mat, the data are stored in a variable called eeg). To attenuate this unwanted interference, we can use a 60-Hz stop-band filter (notch filter): [b,a] = butter(6, [(60-5)/128,(60+5)/128],'stop'). Now type in the following commands:

```
freqz(b,a,100,256)
figure
impz(b,a,100)
figure
plot(eeg)
hold

eegf=filter(b,a,eeg);
plot(eegf,'r')
```

Filter characteristic, its impulse response, and an application are shown in Fig. 13.2.

*Note:* The success of a 60-Hz band reject filter should not be used as an excuse to record poor quality data with lots of hum. First, ideal filters do not exist; therefore, the attenuation is never complete. Second, 50/60-Hz notch filters have a tendency to produce oscillatory artifacts at discontinuities in the signal.

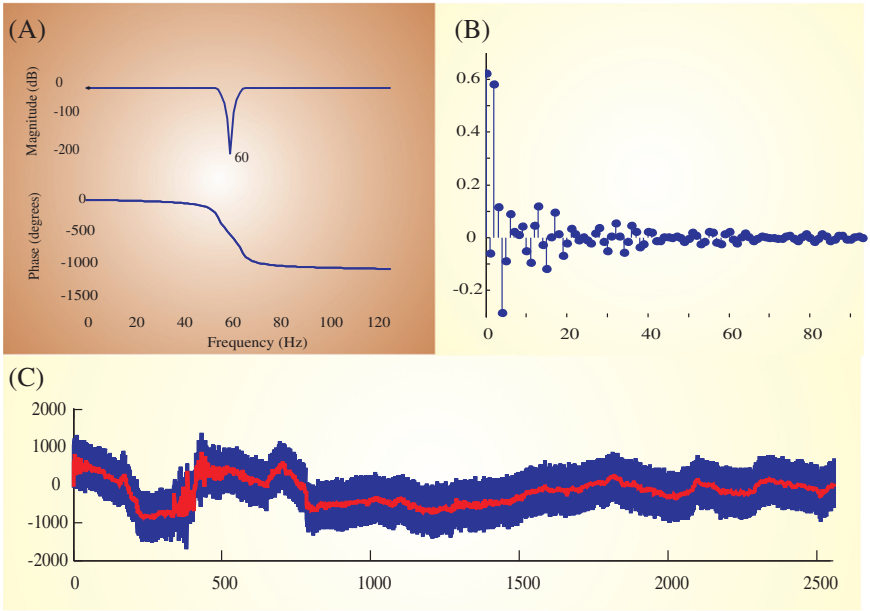



**Figure 13.2** Example of a band-reject filter to remove 60-Hz hum. This type of filter is often called a notch filter. The graphs in (A) represent the Bode plot (the filter characteristic), and (B) shows the first part of the filter's impulse response. In this example, the full impulse response cannot be shown because this is an IIR-type filter. (C) EEG with 60-Hz noise (blue) and the trace that was filtered using the notch filter (red) superimposed.

## 13.6 FILTER TYPES

Thus far we used the Butterworth filter as the basis for most of our analyses. As we saw in the Bode plot (Chapter 12), the characteristics of the Butterworth filter are not ideal; the transition band is fairly wide and the phase response is frequency dependent (e.g., Fig. 12.4). Because the ideal filter cannot be made (Appendix 11.1), we always need to compromise in our approach to the ideal filter characteristic. This compromise may vary with each application. In some cases, strong attenuation of noise is required, but phase response is not critical; in other cases, where we want to accurately measure delays, the phase response is critical. Not surprisingly, in the real world there is a trade-off between a small transition band, also known as a steep roll-off, and a favorable (flat) phase response.

The different filter types realizing different compromises that are available in MATLAB are summarized in Table 13.1. Note that the Butterworth

**Table 13.1** Summary of Roll-off and Phase Characteristics of Different Filter Types That Are Available in the MATLAB Signal Processing Toolbox

| Filter type | MATLAB command | Roll-off | Phase response |
|---|---|---|---|
| Bessel | besself | − | ++ |
| **Butterworth** | butter | ± | ± |
| Chebyshev | cheby1, cheby2 | + | − |
| Elliptic | ellip | ++ | −− |

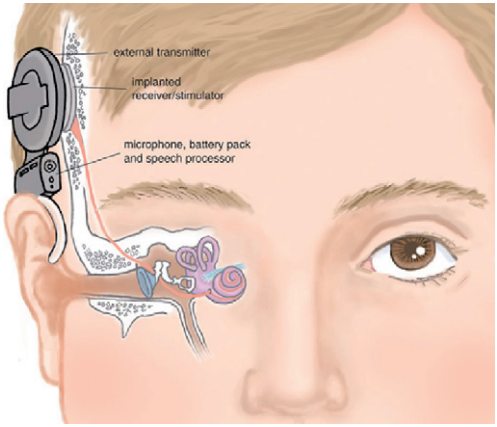
is a good compromise, realizing both a reasonable roll-off and phase response. The Butterworth filter's magnitude response $|H(j\omega)|$ is flat in the pass band and monotonic overall. The Bessel and Elliptic filter types are at the extreme ends of the trade-off scale, realizing either a good phase response or a steep roll-off, respectively. Because Bessel filters are characterized by an almost constant delay for the frequencies across the pass band, they preserve the wave shape of the filtered signal in the time domain. The increased roll-off of the Chebyshev and Elliptic filters comes at the cost of ripple in their magnitude response curves $|H(j\omega)|$. In MATLAB there are cheby1 and cheby2 commands; the type I Chebyshev filter has ripple in the pass band and a flat stop band, type II is the opposite with a flat pass band and ripple in the stop band. The Elliptic filter type has a magnitude response as shown in Figure 10.1 (i.e., ripple in both pass and stop bands).

## 13.7 FILTER BANK

In the previous text, we considered filters with a single input and single output. In some applications, it is beneficial to look at the signal in a set of frequency bands. Instead of a single filter, one constructs a set of filters (a filter bank) with desired frequency responses. The filters in this bank can be applied in parallel to the same signal. This is the preferred approach if you want to explore the signal's frequency content or detect features associated with certain frequency components. As we will see, this approach is also the basis for the so-called spectrogram and scalogram representations of a time series (Chapter 16, Fig. 16.5).

An interesting biomedical application of filter banks is the cochlear implant (Fig. 13.3). This instrument mimics the cochlea by separating the input (sound transduced into an electrical signal by a sensitive microphone) into separate spectral components. Physiologically it is known that the bottom part (base) of the cochlea is more sensitive to

**(A)**




**(B)**

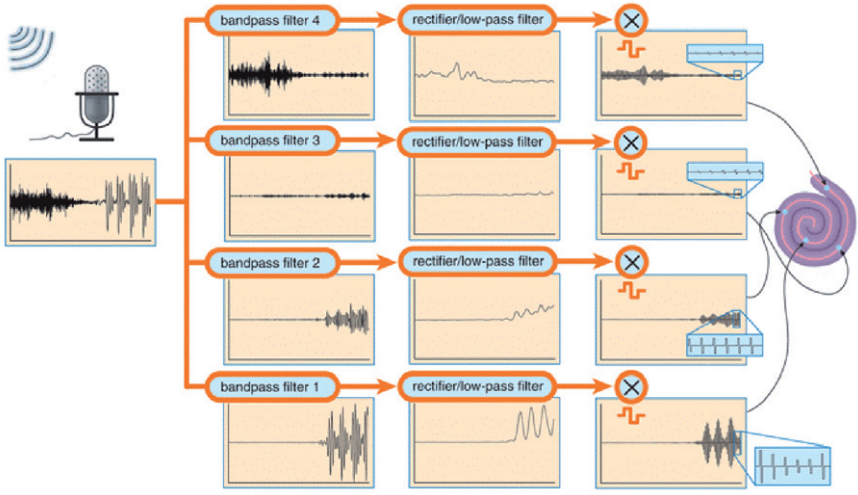



**Figure 13.3** (A) Cochlear implants have five main components, only two of which are inside the body. A microphone above the ear senses sound waves, which are directed to a small computer behind the ear. The computer transforms the signals into a coded stimulus that must be delivered to a set of electrodes that are implanted in the cochlea. These stimulus signals are (via an external transmitter) transmitted through the skin to an implanted receiver, which converts them to electrical stimuli for the implanted electrodes. These stimuli excite the auditory nerve. (B) A diagram of how a filter bank is used to decompose a complex sound signal for the syllable "sa." Band-pass filters 1 to 4 (left column of panels) each pass a specific frequency component of the compound signal in the left panel. Filter 1 passes the lowest frequency components, and filter 4 passes the highest ones. A set of rectifying low-pass filters (middle column of panels) subsequently create an envelope for the activity in each of the frequency bands. This envelope signal is finally transformed into a train of biphasic pulses (right column of panels) that can be used to stimulate a specific location in the cochlea. The high-frequency components stimulate the base of the cochlea, and the low frequencies stimulate nerve fibers more toward the apex. Used with permission from Dorman MF and Wilson BS (2004), The design and function of cochlear implants. *American Scientist* 92: 436–445.

high-frequency components, whereas the top (apex) is more sensitive to low-frequency oscillations. The filter bank in the implant device mimics this normal cochlear operation and stimulates sensors connected to auditory nerve in a pattern analogous to a normal cochlea. Of course, this procedure only works for patients whose auditory system downstream of the cochlea is intact.

## 13.8 FILTERS IN THE SPATIAL DOMAIN

At several instances in the text we noted that our processing techniques on time series $x(t)$ can easily be translated into the spatial domain, such as with an intensity image conceived as a function of two spatial dimensions $I(x,y)$. Here we simply replace the time parameter $t$ in our algorithm by a spatial variable $x$, $y$, or $z$; an example of such an application is described in Chapter 7, Section 7.2.

Spatial filters can be used to remove or enhance spatial frequency components. For instance, a high-pass filter can be used to enhance sudden transitions (edges) in the spatial domain while attenuating slow or gradual changes in the image. An example of such a procedure by using a simple Butterworth filter is shown in Figure 13.4, generated by MATLAB script pr13_1.m. The image of Lena in Figure 13.4A is commonly used to evaluate image processing algorithms because it contains a number of challenging properties that can be enhanced by signal processing techniques and

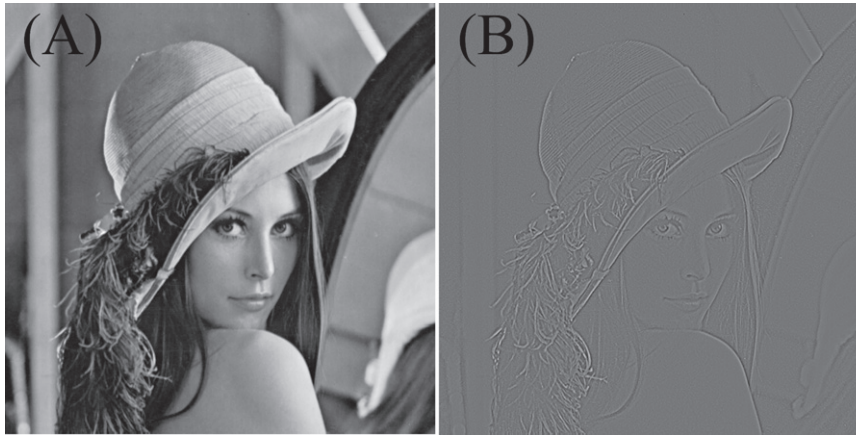



**Figure 13.4** An example of a filter application in the spatial domain using a picture of Lena (A) as input for a two-dimensional Butterworth high-pass filter. Although this spatial filter is not optimized for this application, by using the principle of high-pass filtering we can detect transitions (edges) in the spatial domain as shown in (B).

probably also because the image pleases many male image processing specialists.

***The following is a part of pr13_1.m used to filter input image contained in matrix lena_double:***

```
[b,a]=butter(1,100/256,'high');        % make a high-pass filter based on
                                       % a sample rate of 1 pixel and
                                       % Nyquist of 256 pixels

lenah=lena_double;

for k=1:512;
   lenah(k,:)=filtfilt(b,a,lenah(k,:)); % use filtfilt to prevent phase shift
end;
```

The part of the script shown above successively shows high-pass filters in each horizontal line. The image therefore detects the vertical transitions (edges) in each row. Another part in pr13_1.m detects abrupt horizontal transitions, and the output of both filters can be added to show the edges in the picture; such a result is shown in Figure 13.4B. The detected edges can now be superimposed on the original picture in order to obtain an edge-enhanced image; you can run pr13_1.m to observe these effects. Applications such as the one shown with Lena's image can help you to enhance images but can also be used to detect regions of interest in optical imaging data sets such as microscopic images or movies.

## APPENDIX 13.1

Comparise the butter command in MATLAB with the approximation from Figure 11.3. Using the diagram in Figure 11.3 and the description in section 11.2.2 and Appendix 11.3, we get the following filter equation:

$$\underbrace{1}_{A(1)}\, y_n - \underbrace{\frac{(RC/\Delta t)}{(RC/\Delta t)+1}}_{A(2)} y_{n-1} = \underbrace{\frac{1}{(RC/\Delta t)+1}}_{B} x_n \qquad \text{(A13.1-1)}$$

Using $R = 10^4\Omega$, $C = 3.3\,\mu F$, and $\Delta t = 1/400$, we can calculate the filter coefficients (Equation (13.2)): $A$ = [A(1) A(2)] = [1.0000 –0.9296] and $B$ = [0.0704]. On the other hand, if we used the more precise Equation (A11.3-2), discussed in Appendix 11.3 and repeated here in the same format as Equation (A13.1-1) for convenience:

$$\underbrace{1}_{A(1)}\, y_n \underbrace{-e^{-\Delta t/RC}}_{A(2)} y_{n-1} = \underbrace{(1 - e^{-\Delta t/RC})}_{B} x_n \qquad \text{(A13.1-2)}$$

we get $A$ = [1.0000 –0.9270] and $B$ = [0.0730]. Using a first-order butter command for $f = 1/(2 \times \pi \times R \times C) = 4.8229$ Hz and a sample frequency of 400 Hz (Nyquist frequency: 200 Hz):

$$[B, A] = \text{butter}(1, 4.8229/200)$$
$$B = 0.0365 \quad 0.0365$$
$$A = 1 \quad -0.9270$$

We obtain almost identical values with the difference being that B has two terms, each exactly half of the single term (i.e., 0.0730/2). This has the effect of a moving average of the input.

*Note:* There is also a lot of information about digital filters on the web. For example, www.users.cs.york.ac.uk/~fisher/mkfilter is a really cool website that allows you to specify and design digital filters.