An Introduction to Discrete-Time Filter Design

Michael Rice
Brigham Young University

1 Preliminaries

1.1 Signal Processing for Sampled Data

This laboratory project is devoted to designing filters to operate on sampled data. In these applications, the signal characteristics are described in the continuous time domain. But because the signal processing must be performed in the discrete-time domain, we need to know how the continuous time domain signal properties show up in the discrete-time domain. To quantify the relationship between the Fourier transform of a continuous time signal and the DTFT of its samples, we revisit the discussion of Sections 6-12.4 and 6-12.5, pp. 309–310.

To this end, consider a band limited continuous time signal $x(t)$ with Fourier transform $\hat{\mathbf{X}}(\omega)$. As a conceptual tool, the text defined the impulse train sampled signal $x_s(t)$ defined by

$$x_s(t) = x(t) \times \sum_{n=-\infty}^{\infty} \delta(t - nT_s). \tag{1}$$

Section 6-12.4 derived the Fourier transform $\hat{\mathbf{X}}_s(\omega)$ by leveraging the property that multiplication in the time domain is convolution in the frequency domain:

$$\begin{aligned}
\hat{\mathbf{X}}_s(\omega) &= \frac{1}{2\pi}\hat{\mathbf{X}}(\omega) * \hat{\mathbf{\Lambda}}(\omega) \\
&= \frac{1}{2\pi}\hat{\mathbf{X}}(\omega) * \frac{2\pi}{T_s}\sum_{k=-\infty}^{\infty} \delta\left(\omega - \frac{2\pi k}{T_s}\right) \\
&= \frac{1}{T_s}\sum_{k=-\infty}^{\infty} \hat{\mathbf{X}}\left(\omega - \frac{2\pi k}{T_s}\right).
\end{aligned} \tag{2}$$

This expression teaches us that the Fourier transform of a sampled signal is periodic in ω with period $2\pi/T_s$. Now, the DTFT of $x[n] = x(nT_s)$ is

$$\mathbf{X}(e^{j\Omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\Omega n}. \tag{3}$$

It is not clear from (2) what the relationship is between the DTFT of $x[n]$ and the Fourier transform of $x(t)$. So what is this relationship? To answer this question, we compute the Fourier transform $\hat{\mathbf{X}}_s(\omega)$ directly:

$$\begin{aligned}
\hat{\mathbf{X}}_s(\omega) &= \int_{-\infty}^{\infty} x_s(t) e^{-j\omega t} dt \\
&= \int_{-\infty}^{\infty} x(t) \times \sum_{n=-\infty}^{\infty} \delta(t - nT_s) e^{-j\omega t} dt \\
&= \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} x(nT_s) \delta(t - nT_s) e^{-j\omega t} dt \\
&= \sum_{n=-\infty}^{\infty} x(nT_s) \int_{-\infty}^{\infty} \delta(t - nT_s) e^{-j\omega t} dt \\
&= \sum_{n=-\infty}^{\infty} x(nT_s) e^{-j\omega T_s n} \qquad (4)
\end{aligned}$$

The last line is equal to the DTFT (3) if $\omega T_s = \Omega$. Equating (2) and (4) we have

$$\underbrace{\sum_{n=-\infty}^{\infty} x[n] e^{-j\Omega n}}_{\text{DTFT of } x[n] = x(nT_s)} = \underbrace{\frac{1}{T_s} \sum_{k=-\infty}^{\infty} \hat{\mathbf{X}}\left(\omega - \frac{2\pi k}{T_s}\right)}_{\text{periodic replicas of the Fourier transform of } x(t)}. \qquad (5)$$

This shows that the DTFT of $x[n] = x(nT_s)$ is related to the Fourier transform of $x(t)$ as follows:

> The DTFT of $x[n] = x(nT_s)$ is equal to scaled periodic replicas of the Fourier transform of $x(t)$. The replication period is the sample rate. The amplitude scaling is the sample rate. The discrete-time frequency axis (Ω rads/sample) and the continouous-time frequency axis (ω rads/s) are related by $\omega T_s = \Omega$.

The relationship is illustrated in Figure 1. The figure shows the Fourier transform of ten sinusoids with frequencies 20 kHz, 22 kHz, ..., 38 kHz. These frequencies correspond to the frequencies you will have to handle in the laboratory assignment. The top plot in Figure 1 shows the Fourier transform in the f (cycles/s) variable whereas the second plot in Figure 1 shows the Fourier transform in the ω (rads/s) variable. The third plot is Fourier transform of the impulse sampled signal $x_s(t)$ [see (1)]. Here, the sample rate is $f_s = 100$ ksamples/s (or, $T_s = 10^{-5}$ s/sample). The relationship between the second and third plots is an illustration of the relationship (2). The DTFT of the samples is shown in the fourth plot of Figure 1. Note that the DTFT is periodic in Ω with period

2π as expected. Because of this, it is customary to plot only the 2π interval centered at $\Omega = 0$. The relationship between the third and fourth plots of Figure 1 is an illustration of (4). Finally, the fifth plot of Figure 1 shows the DTFT using a frequency axis scaled to the F (cycles/sample) variable.

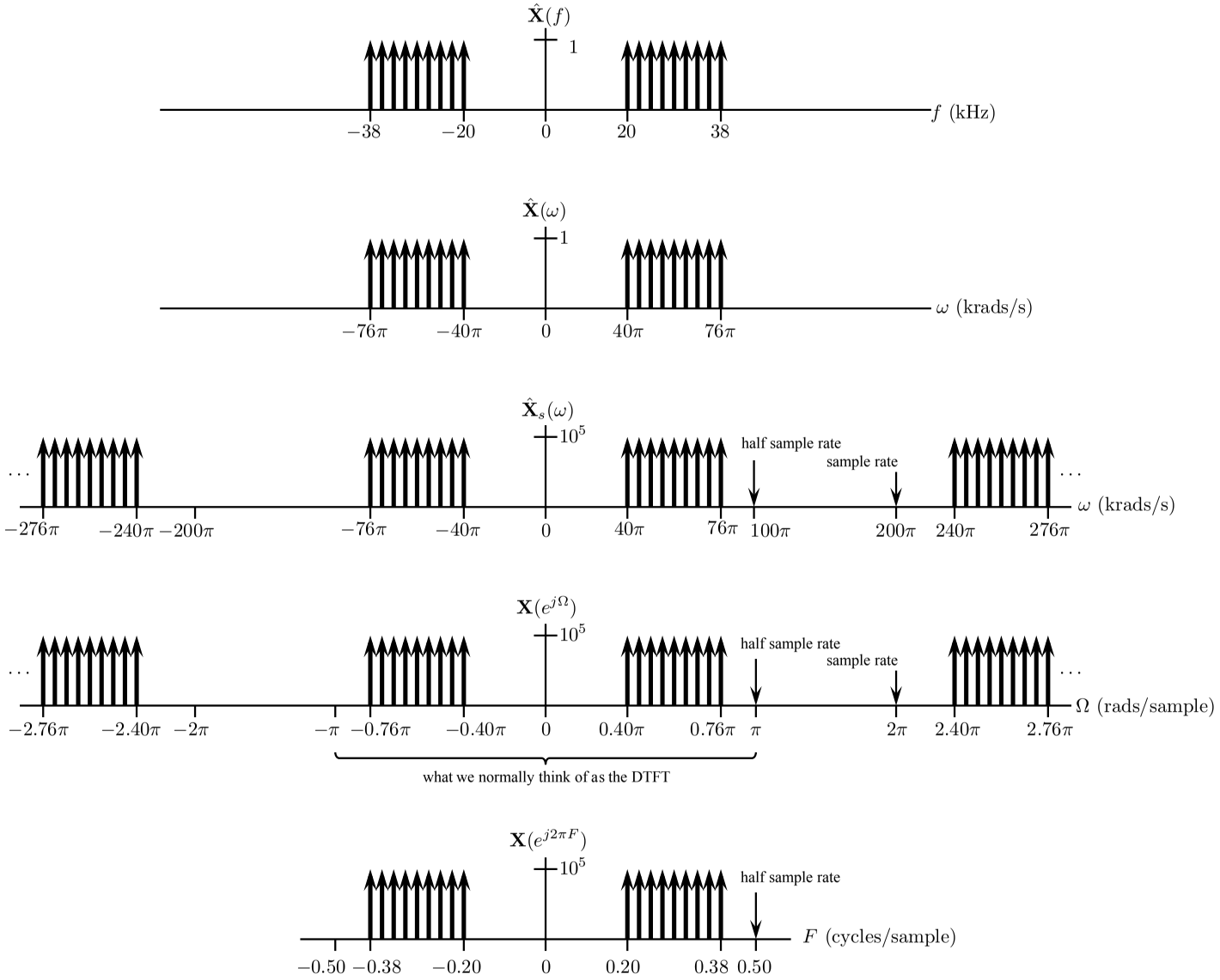


Figure 1: An example illustrating the relationship between the Fourier transform of a continuous time signal and the DTFT of its samples. Here the continuous time signal comprises ten sinusoids. The sample rate is 100 ksamples/s.

1.2 Filter Design

In this laboratory assignment, you will have to "design filters" to detect which of ten frequencies is present the data. Because the phrase "design a filter" is mysterious to junior electrical or computer engineering majors, what this phrase means needs to be defined before moving on.

A convenient starting point is the general $\mathbf{z}$-domain transfer function

$$\mathbf{H}(\mathbf{z}) = \frac{b_0 + b_1\mathbf{z}^{-1} + \cdots + b_{M-1}\mathbf{z}^{-(M-1)}}{1 + a_1\mathbf{z}^{-1} + \cdots + a_{N-1}\mathbf{z}^{-(N-1)}}. \tag{6}$$

This is a *rational function* of $\mathbf{z}$. The word *rational* suggests *ratio*, and ratio refers to the ratio of polynomials. Note that here we follow another convention from the discrete-time signal processing world: we think of rational function as a ratio of polynomials in $\mathbf{z}^{-1}$. In your text, (7.111) is a ratio of polynomials in $\mathbf{z}$. Both forms are equivalent. Both forms are correct. Polynomials in $\mathbf{z}$ are more common in digital control. Following the discrete-time signal processing convention we define the polynomials

$$\begin{aligned} \mathbf{A}(\mathbf{z}) &= 1 + a_1\mathbf{z}^{-1} + \cdots + a_{N-1}\mathbf{z}^{-(N-1)} \\ \mathbf{B}(\mathbf{z}) &= b_0 + b_1\mathbf{z}^{-1} + \cdots + b_{M-1}\mathbf{z}^{-(M-1)} \end{aligned} \tag{7}$$

and think of the $\mathbf{z}$-transform as

$$\mathbf{H}(\mathbf{z}) = \frac{\mathbf{B}(\mathbf{z})}{\mathbf{A}(\mathbf{z})}. \tag{8}$$

The roots of the polynomial $\mathbf{B}(\mathbf{z})$ are called the *zeros* of $\mathbf{H}(\mathbf{z})$ and the roots of the polynomial $\mathbf{A}(\mathbf{z})$ are called the *poles* of $\mathbf{H}(\mathbf{z})$. Here, the LTI system is defined by two lists of numbers

$$\{a_0, a_1, \ldots, a_{N-1}\} \qquad \{b_0, b_1, \ldots, b_{M-1}\}. \tag{9}$$

You have learned that a physically realizable system is one for which $N \geq M$ (see pg. 365 of the text). These two lists of numbers are called the *filter coefficients*. Consequently, an important element of "filter design" involves figuring out what these coefficients need to be to create an LTI system that meets some performance criteria.

In the lab assignment, you will explore two kinds of discrete-time filters: finite impulse response (FIR) filters and infinite impulse response (IIR) filters. These terms, FIR and IRR, refer to the number of non-zero values in the impulse response $h[n]$ of an LTI system. For the purposes of

this lab can think of both of these in terms of the $\mathbf{z}$-domain transfer function

$$\mathbf{H}(\mathbf{z}) = \frac{b_0 + b_1\mathbf{z}^{-1} + \cdots + b_{M-1}\mathbf{z}^{-(M-1)}}{a_0 + a_1\mathbf{z}^{-1} + \cdots + a_{N-1}\mathbf{z}^{-(N-1)}}. \tag{10}$$

The input ($x[n]$), output ($y[n]$) relationship for this filter is the recursion

$$\sum_{i=0}^{N-1} a_i y[n-i] = \sum_{i=0}^{M-1} b_i x[n-i]. \tag{11}$$

[See equation (7.108) on page 364 of the text.]

- FIR filter: the $\mathbf{z}$ domain transfer function of an FIR filter is the special case of $\mathbf{H}(\mathbf{z})$ where $a_0 = 1$ and $a_1 = \cdots = a_{N-1} = 0$:

  $$\mathbf{H}(\mathbf{z}) = b_0 + b_1\mathbf{z}^{-1} + \cdots + b_{M-1}\mathbf{z}^{-(M-1)} \tag{12}$$

  Here, the input-output relationship reduces to

  $$y[n] = \sum_{i=0}^{M-1} b_i x[n-i]. \tag{13}$$

  This is the convolution operation involving the sequence of filter coefficients $\{b_0, b_1, \ldots, b_{M-1}\}$ and the input data sequence $x[n]$. *Caveat lector*! In the next section, you will see the filter coefficients described as the sequence $\{h(-L), \ldots, h(0), \ldots, h(L)\}$. This may appear at first glance to contradict our development. But if we line up the filter coefficients as follows

  $$\begin{array}{cccccccc} b_0 & & b_1 & \cdots & b_{L+1} & \cdots & b_{M-2} & b_{M-1} \\ h(-L) & & h(-L+1) & \cdots & h(0) & \cdots & h(L-1) & h(L), \end{array} \tag{14}$$

  we see that both lists are capable of telling the same story as long a $M = 2L + 1$. From a pole-zero point of view, FIR filter design is equivalent to defining the zeros of the LTI system.

- IIR filter: The class of practical IIR filters is defined by the $\mathbf{z}$-domain transfer function $\mathbf{H}(\mathbf{z})$ where at least one of the a_i for $i > 0$ is not zero. This produces the recursive relationship whose corresponding impulse response goes on for ever. From a pole-zero point of view, IIR filter design is equivalent to defining the poles and zeros of the LTI system.

When we speak of "designing the filter," we mean three things:

1. First, the performance criteria must be defined. In rare cases, the performance criteria are handed to the signal processing engineer. More commonly, the signal processing engineer must derive the performance criteria from a high level description of what is supposed to happen. The most common performance criteria specify the filter passband frequencies and the desired out-of-band attenuation. Additional considerations sometimes come into play, such as maximum allowable passband ripple, linear phase, and computational constraints (i.e., a maximum number of multiplications is allowed.)

2. Equipped with the performance criteria, we must compute the coefficients to produce an LTI system that meets the performance criteria.

3. Lastly, we must instantiate the algorithm. In hardware, this might mean a physical layout in VLSI (for an ASIC) or a VHDL definition (for an FPGA). In software, this means writing C, C++, or Assembly(!) code to compute the recursion (11).

Here, we will focus on the second step and leave the first step and an exercise. Doing the project in MATLAB® renders the third step trivial.

The basic design approaches for FIR and IIR filters are different. Consequently, the two approaches are outlined in two different sections. The natural question for a student to ask is "which is better?" As with most thing, the correct answer is the distressing response,"it depends." But the following are some of the more important considerations:

- Filter order required to meet given filter specifications: Here, "filter order" refers to the length of the FIR filter and to the degree of the denominator polynomial describing the recursive IIR filter. Filter order is important because it determines the number of "multiply-accumulate" operations required to instantiate the filter. The number of "multiply-accumulate" operations defines the complexity of the discrete-time filter. It is almost always the case that the performance specifications can be met by an IIR filter with lower order than an FIR filter.

- Stability: A causal stable LTI system is one for which all of its poles are inside the unit circle. Because an FIR filter has no poles (other than those at the $\mathbf{z}$-plane origin), an FIR filter is always stable (except in the most pathological of cases). An IIR filter, on the other hand, has poles and the potential for stability issues exists. An IIR filter with strict passband and stopband requirements has poles near the unit circle. Coefficient quantization and round-off effects accompanying fixed-point arithmetic may cause these poles to migrate to the wrong side of the unit circle.

- Linear Phase: Many applications require a filter to have linear phase. (Consider taking ECEn 487 for more on what this means.) Linear phase can be guaranteed in an FIR filter by imposing certain symmetry constraints on the filter coefficients. With IIR filters, it is often difficult to produce a linear phase filter, especially at frequencies near the band edge.

- Digital hardware: In digital hardware designs, pipelining is often used to create really really really fast implementations (at the cost of bulk delay through the system). Instantiating an FIR filter with a pipelined architecture is natural — the filter structure itself almost begs the digital designer to pipeline it. IIR filters, on the other hand, are not readily amenable to pipelining. There are some tricks that can be used, but they are application dependent and must be considered on a case-by-case basis.

- Programmable processor: When a filter is to be instantiated on a programmable processor, the dominant factor defining performance is the complexity. Complexity follows directly from filter order. IIR filters, with lower filter order, are usually the better candidates for instantiations in programmable processors assuming linear phase and stability are not important considerations.

These issues are summarized below. The differences between FIR and IIR filters are overstated in some cases (sometimes the poles of the IIR filter are not near the unit circle so that stability is not an issue, sometimes IIR filter phase is "linear enough" to meet the requirements). The exaggerated differences are to help in your initial understanding of the differences between the filter types. Again, those interested in discrete-time filter design should consider taking ECEn 487.

Figure of Merit	FIR	IIR
Filter order	loser	winner
Stability	winner	loser
Linear phase	winner	loser
Pipelined architecture for fast hardware implementation	winner	loser
Software implementation on programmable processor	loser	winner

2 FIR Filter Design

The starting point is the ideal bandpass filter shown in Figure 2 (a). This is a plot of the DTFT of an ideal band-pass filter centered at $\Omega = \Omega_0$ rads/sample with a bandwidth of W rads/sample. That is, we have

$$\mathbf{H}_{\text{ideal}}(e^{j\Omega}) = \begin{cases} 1 & \Omega_0 - \frac{W}{2} \leq |\Omega| \leq \Omega_0 + \frac{W}{2} \\ 0 & \text{otherwise.} \end{cases} \tag{15}$$

The impulse response is inverse DTFT of $\mathbf{H}_{\text{ideal}}(e^{j\Omega})$. The inverse DTFT, given by (7.154b) for $\Omega_1 = -\pi$, is

$$h_{\text{ideal}}[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} \mathbf{H}_{\text{ideal}}(e^{j\Omega}) e^{jn\Omega} d\Omega. \tag{16}$$

Making the appropriate substitutions, we have

$$\begin{aligned} h_{\text{ideal}}[n] &= \frac{1}{2\pi} \int_{-\Omega_0 - W/2}^{-\Omega_0 + W/2} e^{jn\Omega} d\Omega + \frac{1}{2\pi} \int_{\Omega_0 - W/2}^{\Omega_0 + W/2} e^{jn\Omega} d\Omega \\ &= \frac{1}{j2\pi n} e^{jn\Omega} \Big|_{-\Omega_0 - W/2}^{-\Omega_0 + W/2} + \frac{1}{j2\pi n} e^{jn\Omega} \Big|_{\Omega_0 - W/2}^{\Omega_0 + W/2} \\ &= \frac{1}{j2\pi n} \left[e^{jn(-\Omega_0 + W/2)} - e^{jn(-\Omega_0 - W/2)} + e^{jn(\Omega_0 + W/2)} - e^{jn(\Omega_0 - W/2)} \right] \\ &= \frac{1}{j2\pi n} \left[e^{-jn\Omega_0} \left(e^{jnW/2} - e^{-jnW/2} \right) + e^{jn\Omega_0} \left(e^{jnW/2} - e^{-jnW/2} \right) \right] \\ &= \frac{1}{j2\pi n} \left(e^{jnW/2} - e^{-jnW/2} \right) \left(e^{jn\Omega_0} + e^{-jn\Omega_0} \right) \\ &= \frac{2}{\pi n} \sin\left(nW/2\right) \cos\left(n\Omega_0\right) \\ &= \frac{W}{\pi} \frac{\sin\left(nW/2\right)}{nW/2} \cos\left(n\Omega_0\right). \end{aligned} \tag{17}$$

The important observations here are

1. The impulse response of the ideal bandpass filter $h_{\text{ideal}}[n]$ is defined for all $-\infty < n < \infty$. In other words, the ideal bandpass filter is an IIR filter. (But, it is not a recursive filter—think about it.) For a given center frequency Ω_0 and bandwidth W, one can use (17) to compute the filter coefficients.

2. The second term in (17) is a sinc function. The sinc function is defined by equation (5.66) in

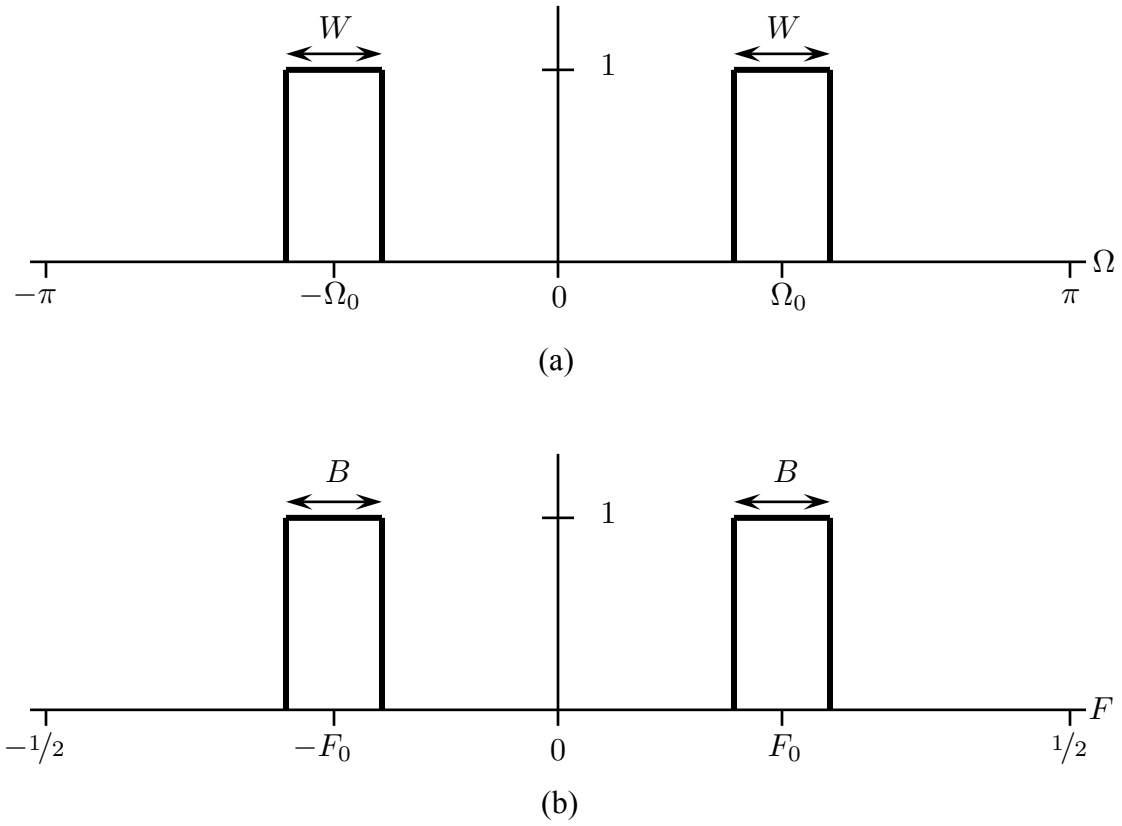


Figure 2: The DTFT of an ideal bandpass filter: (a) The ideal bandpass filter using the traditional traditional Ω-frequency axis. The units for Ω are rads/sample. The bandwidth is W rads/sample and the center frequency is Ω_0 rads/sample. (b) The ideal bandpass filter using a different frequency axis. The units for F are cycles/sample. The bandwidth is B cycles/sample and the center frequency is F_0 cycles/sample. The relationship between the two versions are $\Omega = 2\pi F$, $\Omega_0 = 2\pi F_0$, and $W = 2\pi B$.

Section 5-7 of the text.

3. For the lab project, you may find it more useful to think of the filter parameters in terms of the variables F, F_0 and B shown in Figure 2 (b). Using the relationships $\Omega = 2\pi F$, $\Omega_0 = 2\pi F_0$, and $W = 2\pi B$, (17) may be expressed as

$$h_{\text{ideal}}[n] = 2B \, \frac{\sin(\pi B n)}{\pi B n} \, \cos(2\pi F_0 n). \tag{18}$$

The main problem with the ideal band-pass filter is that it is an IIR filter. We want an FIR filter. A example of (18) for $B = 0.05$ cycles/sample and $F_0 = 0.2$ cycles/sample for $-500 \leq n \leq 500$ is shown in Figure 3. Two important observations apply here and generally:

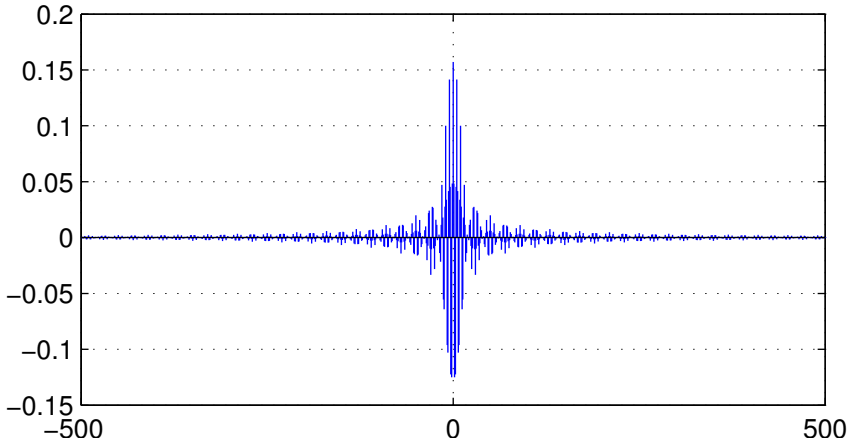


Figure 3: A stem plot of (18) for $B = 0.05$ cycles/sample and $F_0 = 0.2$ cycles/sample for $-500 \leq n \leq 500$.

1. The $|h[n]|$ decreases as $|n|$ increases.

2. The large values of $|h[n]|$ are concentrated in the vicinity of $n = 0$.

These observations suggest a way to create the desired FIR filter from the given IIR filter: *truncate the ideal filter impulse response with the center at* $n = 0$. In mathematical terms, this is

$$h_{\text{FIR}}[n] = \begin{cases} h_{\text{ideal}}[n] & -L \leq n \leq L \\ 0 & \text{otherwise.} \end{cases} \tag{19}$$

The truncation defined by (19) produces a length-$(2L + 1)$ FIR filter. The DTFT of the impulse response (18) for $B = 0.05$ cycles/sample and $F_0 = 0.2$ cycles/sample and for four different truncation lengths is plotted in Figure 4. Here we see that as the length of the filter increases, its DTFT more closely resembles the ideal frequency response.

The frequency-domain plots of Figure 4 also reveal something else: very high sidelobes. Except for the shortest length, notice that the first sidelobe on all of them is about the at about the same level, -20 dB. This means the filter only attenuates the energy in the adjacent band by approximately 20 dB. Is this enough? The answer depends on the application, but usually the answer is *no*.

Clearly, the sidelobes result from the truncation. If we had a way to model the time domain truncation operation in the frequency domain, then we might see how to reduce the sidelobe levels. So, of all the ways to think about truncation, we seek the one that has a useful frequency domain representation. In this light, the best way to think about truncation is as multiplication by a se-

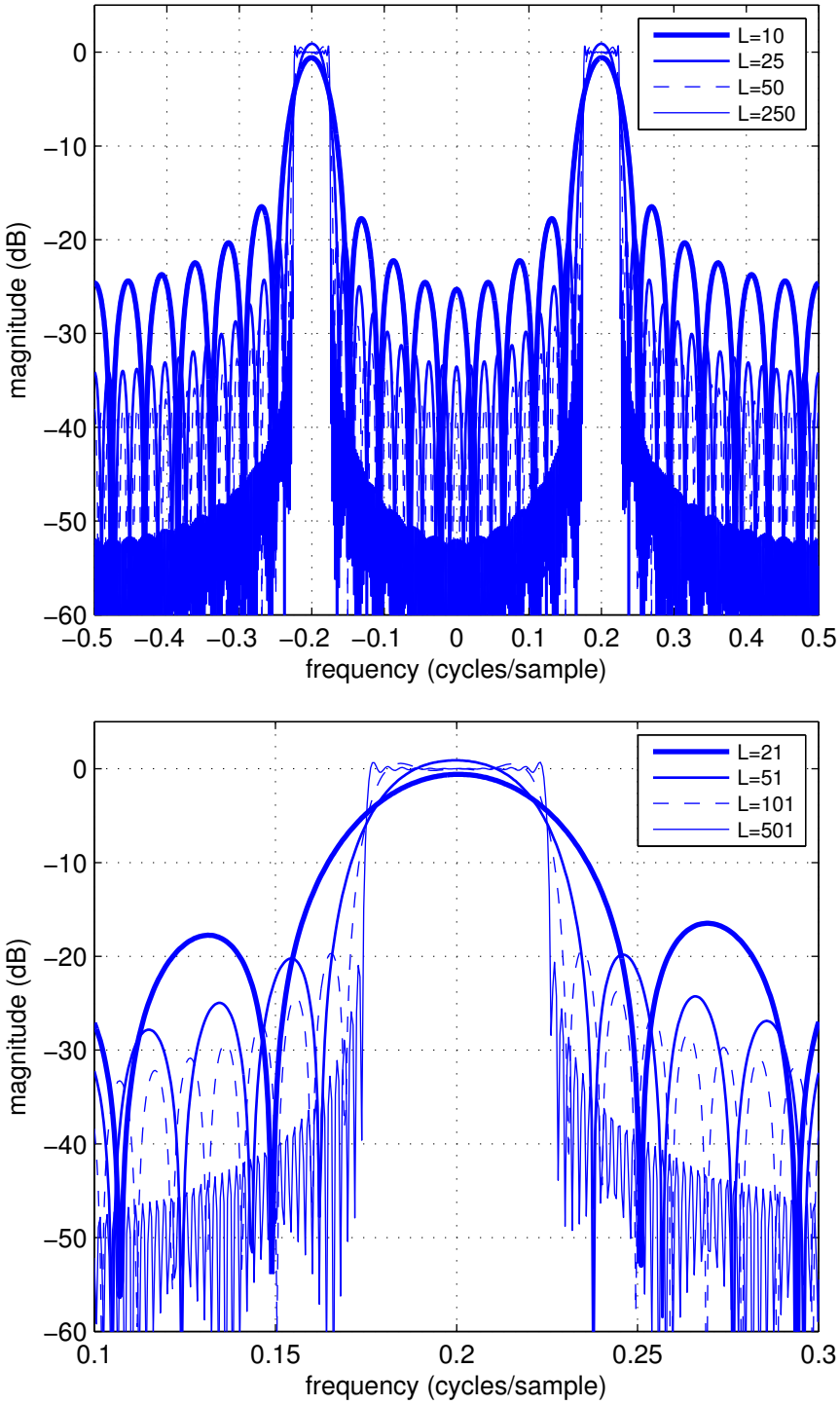


Figure 4: A plot of the DTFT of (18) for $B = 0.05$ cycles/sample and $F_0 = 0.2$ cycles/sample for different truncation lengths. (top) view for $-\pi \leq \Omega < \pi$ rads/sample; (bottom) view for $0.2\pi \leq \Omega < 0.6\pi$ rads/sample.

quence comprising ones in the locations corresponding to the samples we keep, and zeros in the locations corresponding to the samples lost to truncation. The situation is illustrated in Figure 5. Here, the impulse response of the desired FIR filter is

$$h_{\text{FIR}}[n] = w[n] \times h_{\text{ideal}}[n] \tag{20}$$

where the $w[n]$ is the *window*

$$w[n] = \begin{cases} 1 & -L \leq n \leq L \\ 0 & \text{otherwise.} \end{cases} \tag{21}$$

The reason we prefer this representation of time-domain truncation is that we know how to think about it in the frequency domain: multiplication in the time domain is convolution in the frequency domain (see Property 7 in Table 7-7 on page 378):

$$\mathbf{H}_{\text{FIR}}(e^{j\Omega}) = \frac{1}{2\pi}\mathbf{W}(e^{j\Omega}) * \mathbf{H}_{\text{ideal}}(e^{j\Omega}) \tag{22}$$

$$= \frac{1}{2\pi}\int_{-\pi}^{\pi} \mathbf{W}(e^{j(\lambda-\Omega)})\mathbf{H}_{\text{ideal}}(e^{j\lambda})d\lambda. \tag{23}$$

The DTFT of the window is

$$\mathbf{W}(e^{j\Omega}) = \sum_{n=-L}^{L} e^{-j\Omega n} = \frac{\sin\left((2L+1)\dfrac{\Omega}{2}\right)}{\sin\left(\dfrac{\Omega}{2}\right)}. \tag{24}$$

The frequency-domain convolution is illustrated in Figure 6. The figure shows that the sidelobes observed in $\mathbf{H}_{\text{FIR}}(e^{j\Omega})$ are a result of the sidelobes in $\mathbf{W}(e^{j\Omega})$. The important observations here are

1. The closer $\mathbf{W}(e^{j\Omega})$ to an impulse, the closer $\mathbf{H}_{\text{FIR}}(e^{j\Omega})$ is to $\mathbf{H}_{\text{ideal}}(e^{j\Omega})$. The only way to make $\mathbf{W}(e^{j\Omega})$ close to an impulse is to make L large. This explains the plots in Figure 4: as the length increased, $\mathbf{W}(e^{j\Omega})$ more closely approximated an impulse and $\mathbf{H}_{\text{FIR}}(e^{j\Omega})$ more closely approximated $\mathbf{H}_{\text{ideal}}(e^{j\Omega})$.

2. (This one is the mind blower.) We could use *another function* to window $h_{\text{ideal}}[n]$. A good window has a narrow main lobe and low side lobes. (That is, it must look like an impulse!) For a finite-length sequence, narrow main lobe and low sidelobes are competing demands. So, we must accept a tradeoff. But in general, we prefer windows that are smooth and avoid

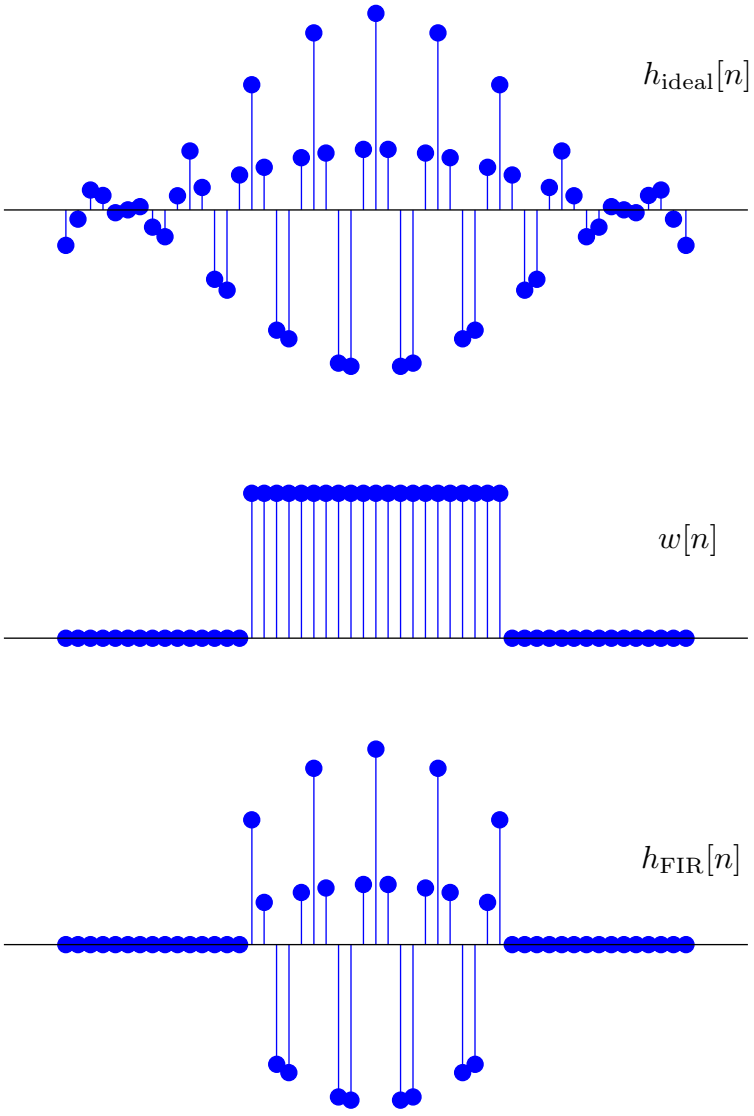


Figure 5: A graphical illustration of modeling time domain truncation as multiplication by the window function $w[n]$.

Table 1: The windows implemented by MATLAB®.

MATLAB® command	Description
bartlett(N)	the Bartlett window
barthannwin(N)	the modified Bartlett-Hann window
blackman(N)	the Blackman window
blackmanharris(N)	the minimum 4-term Blackman-Harris window
bohmanwin(N)	the Bohman window
chebwin(N,R)	the Chebyshev window
flattopwin(N)	the flat top window
gausswin(N,alpha)	the Gaussian window
hamming(N)	the Hamming window
hann(N)	the Hann window
kaiser(N,Beta)	the Kaiser window
nuttallwin(N)	the Nutall modified minimum 4-term Blackman-Harris window
parzenwin(N)	the Parzen de la Valle-Poussin window
rectwin(N)	the rectangular window
taylorwin(N,NBAR,SLL)	the Taylor window
tukeywin(N,R)	the Tukey window
triang(N)	the triangular window

the sharp corners of the rectangular window. There are a number of windows that represent different points in the tradeoff space. MATLAB® implements the windows listed in Table 1.

An example using the Blackman-Harris window is shown in Figure 7. Here, $h_{\text{ideal}}[n]$ is given by (18) for $B = 0.05$ cycles/sample and $F_0 = 0.2$ cycles/sample. The frequency responses are for the windowed versions of (18) using the length-101 rectangular and Blackman-Harris windows. Observe that with the Blackman-Harris window, the sidelobes are much much lower, but the main lobe (the pass band) is wider. The MATLAB® code that generates the plot is the following:

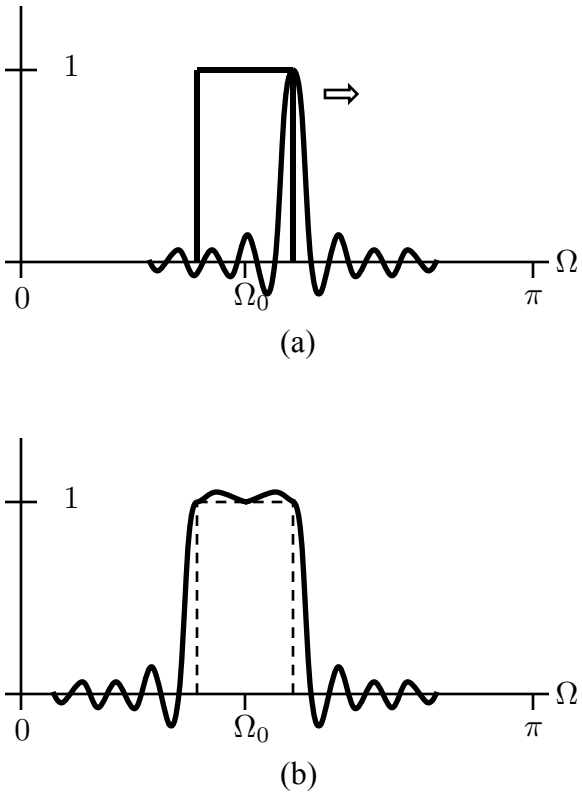


Figure 6: A graphical representation of the frequency-domain convolution describing windowing (truncation) in the time domain. Note that only the positive frequency axis is shown — this is to keep the illustration simple.

```
F0 = 0.20;                          % center frequency (cycles/sample)
B = 0.05;                           % bandwidith (cycles/sample)

L = 51;                             % the length parameter L
N = 2*L+1;                          % the filter length
n = (-L:L)';                        % the sample index
hideal = 2*B*cos(2*pi*F0*n).*sinc(B*n);
h0 = rectwin(N).*hideal;            % window with rectangular window
h1 = blackmanharris(N).*hideal      % window with Blackman-Harris window

Nd = 2048;                          % number of points around unit circle
FF = -0.5:1/Nd:0.5-1/Nd;            % frequency axis for DFT plots
H0 = freqz(h0,1,Nd,'whole');        % DFT of h0
H1 = freqz(h1,1,Nd,'whole');        % DFT of h1

% plot H0 and H1 in dB
% and zoom in on the passband

plot(FF,20*log10(abs(fftshift(H0))),'b-',FF,20*log10(abs(fftshift(H1))),'b--');
grid on;
xlabel('frequency (cycles/sample)');
ylabel('magnitude (dB)');
axis([0.1 0.30 -60 5]);
set(gca,'XTick',[0.1 0.15 0.2 0.25 0.3]);
legend('retangular window','Blackman-Harris window');
```

MATLAB® code notes:

1. Compare the 7th line (`hideal = ···`) with equation (18). Equation (18) includes the term $\sin(\pi B)/(\pi B)$ which your textbook would call $\text{sinc}(\pi B)$; see (5.66) on page 213. In contrast, line 7 includes the term $\text{sinc}(B)$—the π is missing. This is because MATLAB® uses the definition $\text{sinc}(x) = \sin(\pi x)/(\pi x)$; see the footnote on page 213.

2. The MATLAB® code uses the built in functions `rectwin` and `blackmanharris` to compute the window functions.

3. The function `freqz(b,a,Nd,'whole')` evaluates

$$\mathbf{H}(\mathbf{z}) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_M z^{-M}}{1 + a_1 z^{-1} + a_2 z^{-2} + a_N z^{-N}} \tag{25}$$

at $\mathbf{z} = \exp(j2\pi k/N_d)$ for $k = 0, 1, \ldots, N_d - 1$. Here, the inputs b and a are the vectors

$$b = [b_0, b_1, \ldots, b_M] \qquad a = [1, a_1, a_2, \ldots, a_N]. \tag{26}$$

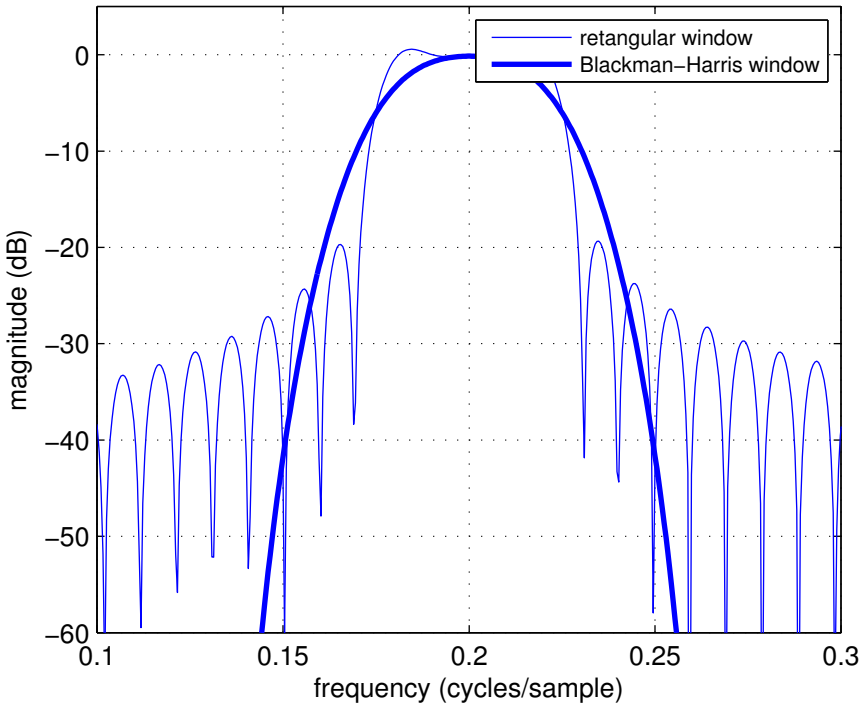


Figure 7: The effect on using different windows. The ideal impulse response is given by (18) for $B = 0.05$ cycles/sample and $F_0 = 0.2$ cycles/sample. The frequency responses are for the windowed versions of (18). The length of the window is 101.

In words, the function returns the $\mathbf{H}(\mathbf{z})$ evaluated at N_d equally spaced points on the unit circle. This is a discretized (in Ω) version of the operation described in Section 8-1.1 of the text. Thus, `freqz` returns samples of the DTFT $\mathbf{H}(e^{j\Omega})$ at $\Omega = 2\pi/N_d \times k$ for $k = 0, 1, \ldots, N_d - 1$. In this case, we have $a_1 = a_2 = \cdots = a_N = 0$, so the second argument of `freqz()` is 1. The ordering of the outputs corresponds to

$$\Omega = 0, 2\pi/N_d, 4\pi/N_d, \ldots, 2\pi(N_d - 1)/N_d.$$

Because we like to plot the DTFT in the range $-\pi \leq \Omega < \pi$ instead of $0 \leq \Omega < 2\pi$, the MATLAB® function `fftshift` is used to reorder the samples for the plot. In other words, `fftshift(H0)` orders the elements of `H0` to match that of the frequency samples in the vector `FF`.

3 Recursive Filter Design

Discrete-time recursive filters are described by their $\mathbf{z}$-domain transfer functions. The $\mathbf{z}$-transfer function of a recursive system is given by (7.111):

$$\mathbf{H}(\mathbf{z}) = \frac{\mathbf{Y}(\mathbf{z})}{\mathbf{X}(\mathbf{z})} = \frac{b_0 + b_1\mathbf{z}^{-1} + \cdots + b_{M-1}\mathbf{z}^{-(M-1)}}{1 + a_1\mathbf{z}^{-1} + \cdots + a_{N-1}\mathbf{z}^{-(N-1)}} \tag{27}$$

This is a *rational function* of $\mathbf{z}$. The word *rational* suggests *ratio*, and ratio refers to the ratio of polynomials. Note that here we follow another convention from the discrete-time signal processing world: we think of rational function as a ratio of polynomials in $\mathbf{z}^{-1}$. In your text, (7.111) is a ratio of polynomials in $\mathbf{z}$. Both forms are equivalent. Both forms are correct. Polynomials in $\mathbf{z}$ are more common in digital control. Following the discrete-time signal processing convention we defined the polynomials

$$\begin{aligned} \mathbf{A}(\mathbf{z}) &= 1 + a_1\mathbf{z}^{-1} + \cdots + a_{N-1}\mathbf{z}^{-(N-1)} \\ \mathbf{B}(\mathbf{z}) &= b_0 + b_1\mathbf{z}^{-1} + \cdots + b_{M-1}\mathbf{z}^{-(M-1)} \end{aligned} \tag{28}$$

and think of the $\mathbf{z}$-transform as

$$\mathbf{H}(\mathbf{z}) = \frac{\mathbf{B}(\mathbf{z})}{\mathbf{A}(\mathbf{z})}. \tag{29}$$

The roots of the polynomial $\mathbf{B}(\mathbf{z})$ are called the *zeros* of $\mathbf{H}(\mathbf{z})$ and the roots of the polynomial $\mathbf{A}(\mathbf{z})$ are called the *poles* of $\mathbf{H}(\mathbf{z})$.

Designing a recursive filter means defining the locations of the poles and zeros of $\mathbf{H}(\mathbf{z})$ so that $\mathbf{H}(e^{j\Omega})$ meets the performance specifications. This is one of the challenging concepts for those new to filter design: the performance specifications define the desired properties for the DTFT $\mathbf{H}(e^{j\Omega})$; but the design occurs in the z-plane defined by $\mathbf{H}(\mathbf{z})$. As long as one remembers two things, this approach is not so bad:

1. $\mathbf{H}(e^{j\Omega}) = \mathbf{H}(\mathbf{z})$ where $\mathbf{z} = e^{j\Omega}$.

2. A graphical method for the relationship between the poles and zeros of $\mathbf{H}(\mathbf{z})$ and the magnitude and phase of $\mathbf{H}(e^{j\Omega})$ is described in Section 8-1 of your text.

The role of pole placement to design a recursive bandpass filter is well illustrated by the following example. Consider the simple single pole system given by

$$\mathbf{H}(\mathbf{z}) = \frac{1}{1 - \mathbf{z}^{-1}}.$$

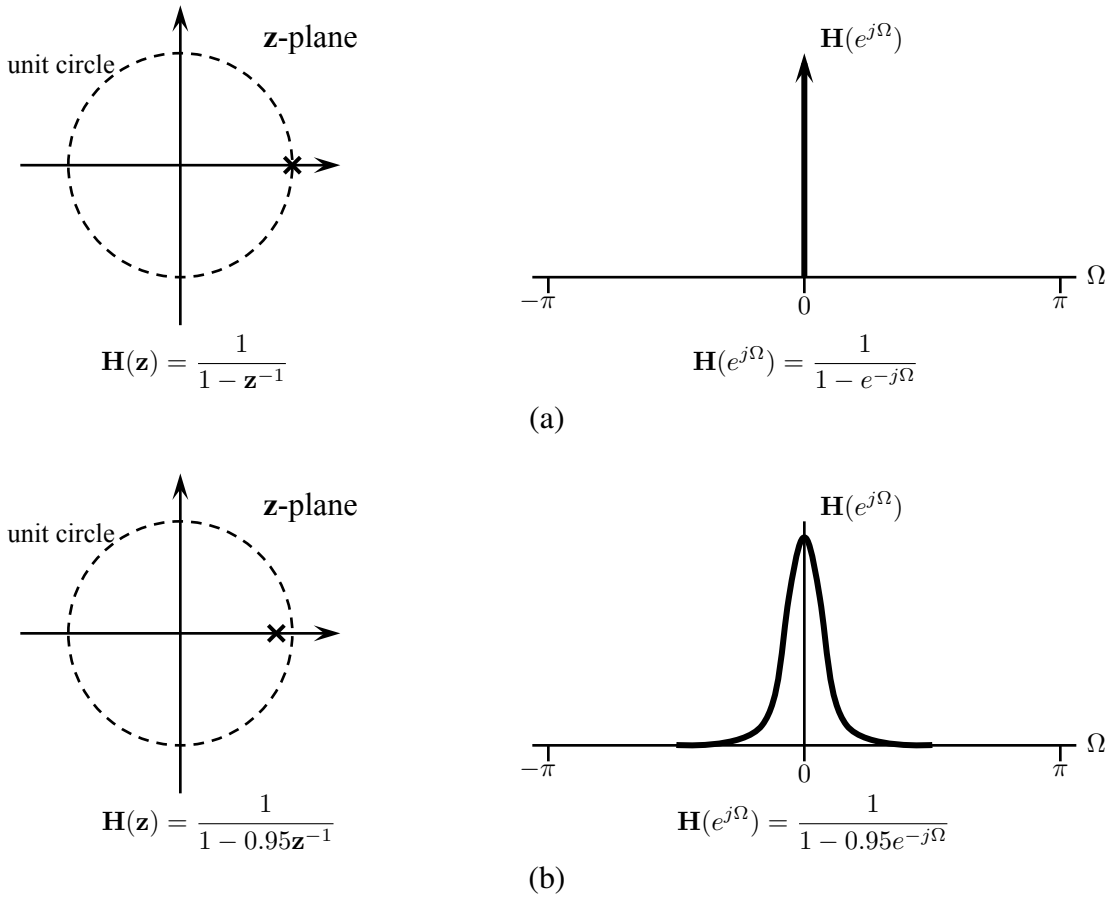


Figure 8: A single pole low-pass filter: (a) the lowpass filter with a pole at $\mathbf{z} = 1$; (b) the lowpass filter with a pole at $\mathbf{z} = 0.95$.

This system has one pole at $\mathbf{z} = 1$ as illustrated in Figure 8 (a). The corresponding DTFT is

$$\mathbf{H}(e^{j\Omega}) = \frac{1}{1 - e^{-j\Omega}}$$

and is also shown Figure 8 (a). Because the pole is on the unit circle, the DTFT comprises an impulse at $\Omega = 0$ rads/sample as shown. This is not a very useful filter. (Is it stable?) To create a usable lowpass filter the pole must be moved just off the unit circle. Should we move it just inside or outside the unit circle? In Figure 8 (b) the pole is moved from $\mathbf{z} = 1$ to $\mathbf{z} = 0.95$. (Why did we move the pole *inside* the unit circle?) The corresponding DTFT is also shown in Figure 8 (b).

Following the discussion of Section 8-1.4, a bandpass filter with passband centered at $\Omega = \Omega_0$ is a system with a pole close to unit circle at an angle Ω_0 relative to the positive real $\mathbf{z}$ axis. This

means a bandpass filter can be created from the lowpass filter of Figure 8 (b) by rotating the pole by Ω_0. The pole is rotated by multiplying it by $e^{j\Omega_0}$. The result is

$$\mathbf{H}_1(\mathbf{z}) = \frac{1}{1 - 0.95e^{j\Omega_0}\mathbf{z}^{-1}}$$

and is shown in Figure 9 (a). The corresponding DTFT, also shown in Figure 9 (a), displays a passband centered at $\Omega = \Omega_0$. Because the DTFT is not symmetric, the filter impulse response (given by the inverse DTFT) is complex-valued. To create a real-valued filter, we require a DTFT displays complex-conjugate symmetry in Ω. The first step in creating a bandpass filter with the required symmetry is to create a second bandpass filter centered at $\Omega = -\Omega_0$. This filter is

$$\mathbf{H}_2(\mathbf{z}) = \frac{1}{1 - 0.95e^{-j\Omega_0}\mathbf{z}^{-1}}$$

and is shown in Figure 9 (b). The corresponding DTFT, also shown in Figure 9 (b), displays a passband centered at $\Omega = -\Omega_0$. The desired filter is the sum of these two filters:

$$\mathbf{H}(\mathbf{z}) = \mathbf{H}_1(\mathbf{z}) + \mathbf{H}_2(\mathbf{z}).$$

The result is shown in Figure 9 (c). Note that because

$$\begin{aligned}
\mathbf{H}(\mathbf{z}) &= \frac{1}{1 - 0.95e^{j\Omega_0}\mathbf{z}^{-1}} + \frac{1}{1 - 0.95e^{-j\Omega_0}\mathbf{z}^{-1}} \\
&= \frac{1 - 0.95e^{-j\Omega_0}\mathbf{z}^{-1} + 1 - 0.95e^{j\Omega_0}\mathbf{z}^{-1}}{(1 - 0.95e^{j\Omega_0}\mathbf{z}^{-1})(1 - 0.95e^{-j\Omega_0}\mathbf{z}^{-1})} \\
&= \frac{1 - 1.9\cos(\Omega_0)\mathbf{z}^{-1}}{1 - 1.9\cos(\Omega_0)\mathbf{z}^{-1} + 0.9025\mathbf{z}^{-2}},
\end{aligned}$$

$\mathbf{H}(\mathbf{z})$ has a zero at $\mathbf{z} = 1.9\cos(\Omega_0)$. This zero, shown in the pole-zero plot of Figure 9 (c), forces $\mathbf{H}(e^{j\Omega})$ to be small at $\Omega \approx 0$ as shown in DTFT plot of Figure 9 (c).

This approached can be generalized to create a recursive bandpass filter based on an a recursive low-pass filter. Let

$$\mathbf{H}_{\text{LPF}}(\mathbf{z}) = \frac{\mathbf{B}(\mathbf{z})}{\mathbf{A}(\mathbf{z})} \tag{30}$$

be the $\mathbf{z}$ transform of an n-th order recursive lowpass filter. For now, assume the degrees of $\mathbf{A}(\mathbf{z})$

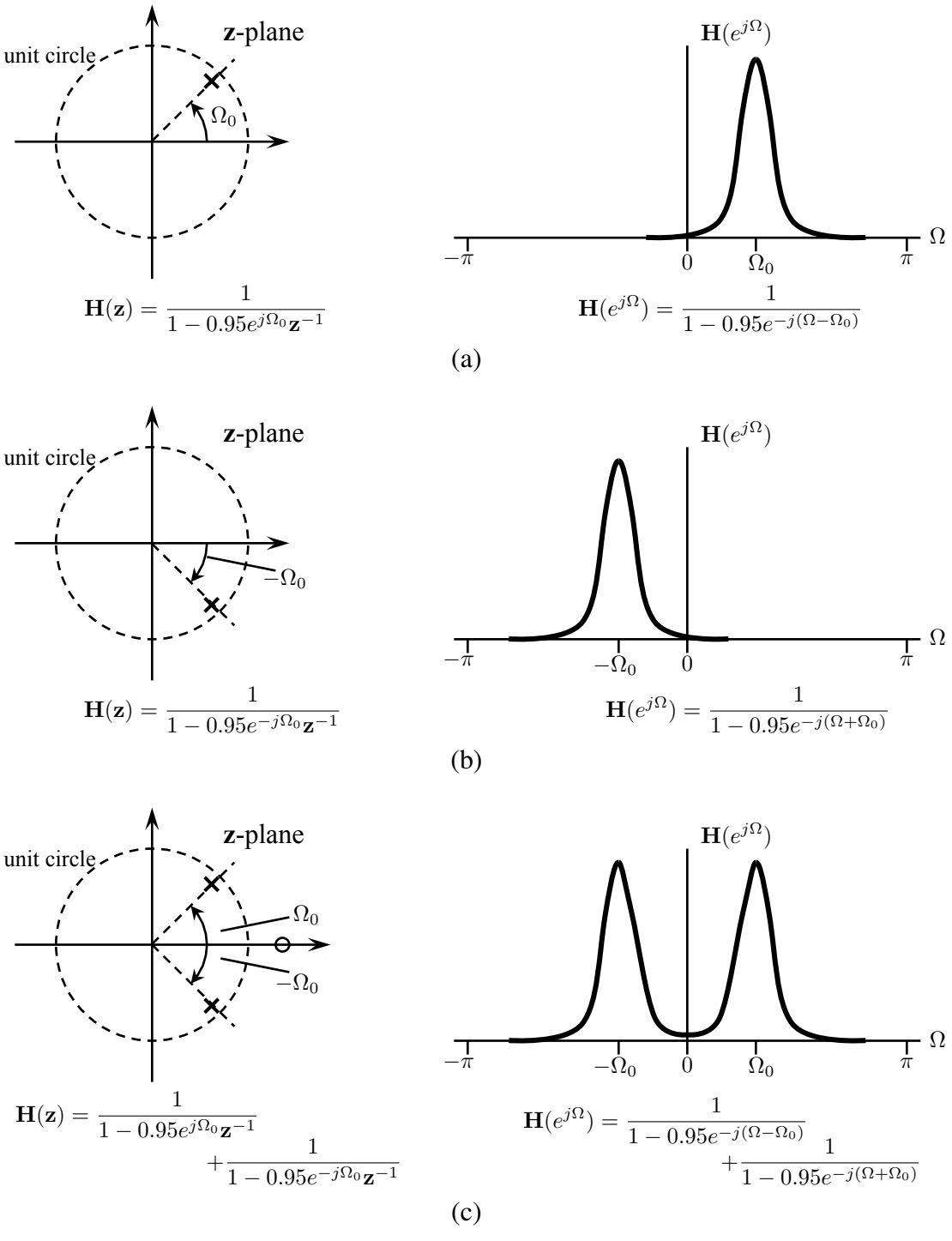


Figure 9: A single pole band-pass filter: (a) the complex-valued bandpass filter with a pole at $\mathbf{z} = 0.95e^{j\Omega_0}$; (b) the complex-valued bandpass filter with a pole at $\mathbf{z} = 0.95e^{-j\Omega_0}$; (c) the real-valued bandpass filter created from the first two filters.

and $\mathbf{B}(\mathbf{z})$ are both n:

$$\mathbf{A}(\mathbf{z}) = 1 + a_1\mathbf{z}^{-1} + \cdots + a_n\mathbf{z}^{-n} \tag{31}$$

$$\mathbf{B}(\mathbf{z}) = b_0 + b_1\mathbf{z}^{-1} + \cdots + b_n\mathbf{z}^{-n}. \tag{32}$$

To rotate the poles of $\mathbf{H}_{\text{LPF}}(\mathbf{z})$ by Ω_0, each root of $\mathbf{A}(\mathbf{z})$ must be multiplied by $e^{j\Omega_0}$. To do this, we rewrite $\mathbf{A}(\mathbf{z})$ in terms of its roots:

$$\mathbf{A}(\mathbf{z}) = (1 - \rho_1\mathbf{z}^{-1})(1 - \rho_2\mathbf{z}^{-1}) \cdots (1 - \rho_n\mathbf{z}^{-1}) \tag{33}$$

where $\rho_1, \rho_2, \ldots, \rho_n$ are the n roots of $\mathbf{A}(\mathbf{z})$. Multiplying each pole by $e^{j\Omega_0}$ produces

$$\mathbf{A}^+(\mathbf{z}) = (1 - \rho_1 e^{j\Omega_0}\mathbf{z}^{-1})(1 - \rho_2 e^{j\Omega_0}\mathbf{z}^{-1}) \cdots (1 - \rho_n e^{j\Omega_0}\mathbf{z}^{-1}). \tag{34}$$

Multiplying this out and collecting terms with common power of $\mathbf{z}^{-1}$ gives

$$\mathbf{A}^+(\mathbf{z}) = 1 + a_1 e^{j\Omega_0}\mathbf{z}^{-1} + a_2 e^{j2\Omega_0}\mathbf{z}^{-2} + \cdots + a_n e^{jn\Omega_0}\mathbf{z}^{-n}. \tag{35}$$

To rotate the zeros of $\mathbf{H}_{\text{LPF}}(\mathbf{z})$ by Ω_0, we multiply the roots of $\mathbf{B}(\mathbf{z})$ by $e^{j\Omega_0}$. The result is

$$\mathbf{B}^+(\mathbf{z}) = b_0 + b_1 e^{j\Omega_0}\mathbf{z}^{-1} + b_2 e^{j2\Omega_0}\mathbf{z}^{-2} + \cdots + b_n e^{jn\Omega_0}\mathbf{z}^{-n}. \tag{36}$$

Following the same procedure, the poles and zeros of $\mathbf{H}_{\text{LPF}}(\mathbf{z})$ are rotated by $-\Omega_0$ by multiplying them by $e^{-j\Omega_0}$ to produce

$$\mathbf{A}^-(\mathbf{z}) = 1 + a_1 e^{-j\Omega_0}\mathbf{z}^{-1} + a_2 e^{-j2\Omega_0}\mathbf{z}^{-2} + \cdots + a_n e^{-jn\Omega_0}\mathbf{z}^{-n} \tag{37}$$

$$\mathbf{B}^-(\mathbf{z}) = b_0 + b_1 e^{-j\Omega_0}\mathbf{z}^{-1} + b_2 e^{-j2\Omega_0}\mathbf{z}^{-2} + \cdots + b_n e^{-jn\Omega_0}\mathbf{z}^{-n}. \tag{38}$$

The desired bandpass filter is defined by the transfer function

$$\mathbf{H}_{\text{BPF}}(\mathbf{z}) = \frac{\mathbf{B}^+(\mathbf{z})}{\mathbf{A}^+(\mathbf{z})} + \frac{\mathbf{B}^-(\mathbf{z})}{\mathbf{A}^-(\mathbf{z})} = \frac{\mathbf{B}^+(\mathbf{z})\mathbf{A}^-(\mathbf{z}) + \mathbf{B}^-(\mathbf{z})\mathbf{A}^+(\mathbf{z})}{\mathbf{A}^+(\mathbf{z})\mathbf{A}^-(\mathbf{z})}. \tag{39}$$

The following segment of MATLAB® code creates a 6-th order bandpass filter from a 3-rd order Butterworth lowpass filter. The code computes the polynomial products required in (39) by exploiting the fact that the coefficients of the product of two polynomials is given by the convolution of the coefficients of multiplicand and multiplier polynomials.

```
n = 3;                                          % LPF filter order
Wc = 2*pi*0.02;                                 % LPF corner frequency
W0 = 2*pi*0.1;                                  % BPF center frequency
[b,a] = butter(n,Wc/pi);                        % create 3rd order B'worth LPF

aplus = a.*exp(1i*W0*(0:n));                    % rotate poles by W0
bplus = b.*exp(1i*W0*(0:n));                    % rotate zeros by W0
aminus = a.*exp(-1i*W0*(0:n));                  % rotate poles by -W0
bminus = b.*exp(-1i*W0*(0:n));                  % rotate zeros by -W0

bb = conv(bplus,aminus) + conv(bminus,aplus);   % BPF zeros
aa = conv(aplus,aminus);                        % BPF poles
aa = real(aa);                                  % eliminate round-off error

figure(1);
subplot(211);
zplane(b,a);                                    % pole-zero plot of LPF
axis(1.2*[-1 1 -1 1]);
subplot(212);
zplane(bb,aa);                                  % pole-zero plot of BPF
axis(1.2*[-1 1 -1 1]);

N = 1024;                                       % # points on unit circle
FF = -0.5:1/N:0.5-1/N;                          % corresponding freq. axis
H_lpf = freqz(b,a,N,'whole');                   % DFT of LPF
H_bpf = freqz(bb,aa,N,'whole');                 % DFT of BPF

figure(2);
subplot(211);
plot(FF,20*log10(abs(fftshift(H_lpf))));        % plot DFT of LPF
grid on;
xlabel('frequency (cycles/sample)');
ylabel('magnitude (dB)');
set(gca,'XTick',-0.5:0.1:0.5);
axis([-0.5 0.5 -60 3]);

subplot(212);
plot(FF,20*log10(abs(fftshift(H_bpf))));        % plot DFT of BPF
grid on;
xlabel('frequency (cycles/sample)');
ylabel('magnitude (dB)');
set(gca,'XTick',-0.5:0.1:0.5);
axis([-0.5 0.5 -60 3]);
```

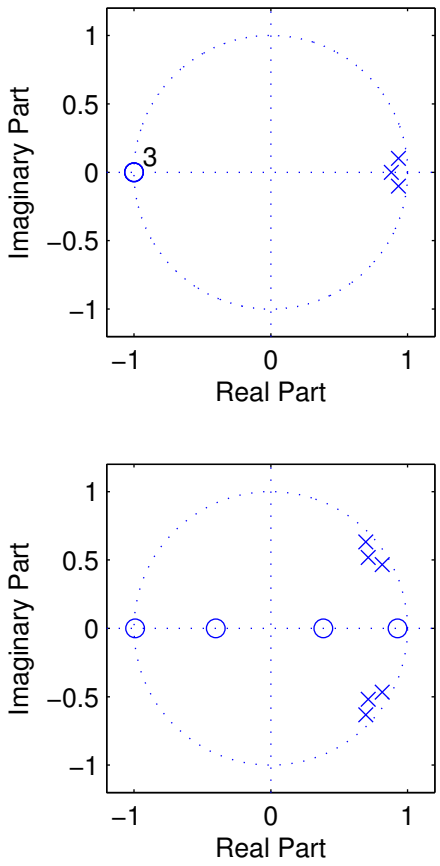


Figure 10: Pole-zero plots: (top) the 3-rd order Butterworth lowpass filter; (bottom) the 6-th order bandpass filter crated from the lowpass filter.

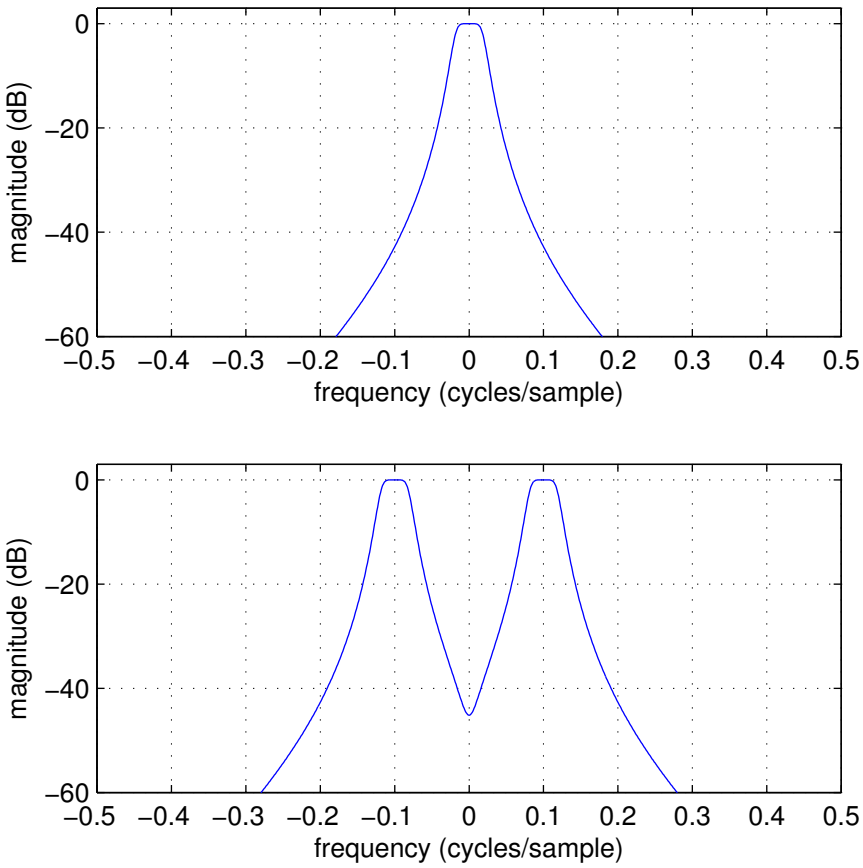


Figure 11: Frequency domain transfer functions: (top) the 3rd-order Butterworth lowpass filter; (bottom) the 6-th order bandpass filter created from the lowpass filter. Cf. Figure 10.

A Some Background on IIR Filter Design

In ECEn 340, students are required to design and construct a 2nd-order Butterworth filter. This filter is used as an anti-aliasing filter on the input side to an analog-to-digital (A/D) converter. (See Section 6-12.12 for a discussion on the need for an anti-aliasing filter prior to sampling.) The filter requirements are derived from the application:

1. Because the sample rate is 100 ksamples/s, the filter bandwidth must be 50 kHz. For a Butterworth filter, the bandwidth is defined as "corner frequency." The corner frequency is the frequency ω_c where $|\hat{\mathbf{H}}(\omega_c)|^2$ is $1/2$ its peak value. Because $10\log_{10}(1/2) = -3$ dB, the corner frequency is sometimes called the "3-dB frequency." See Figure 6-2 (a), pg. 247. So, the filter requirement is $\omega_c = 100\pi$ krads/s.

2. The order of the filter (this will be explained below) is derived from complexity considerations: a 2nd-order system is relatively easy to implement and there is not enough circuit board area to do much more.

The procedure used by the ECEn 340 students was the following:

1. The s-domain transfer function of an n-th order Butterworth lowpass filter is

$$\mathbf{H}(\mathbf{s}) = \frac{1}{\mathbf{D}_n(\mathbf{s})}$$

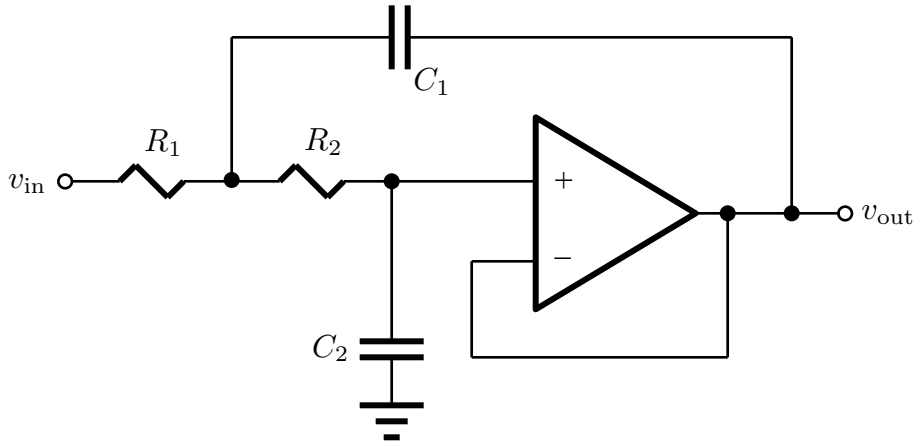
where $\mathbf{D}_n(\mathbf{s})$ is the n-th order Butterworth polynomial. A list of these polynomials for $n = 1, 2, \ldots, 10$ is listed in Table 6-3 on page 285 of the text. Here we have $\mathbf{D}_2(\mathbf{s}) = \mathbf{s}^2 + \sqrt{2}\mathbf{s} + 1$ so that

$$\mathbf{H}(\mathbf{s}) = \frac{1}{\mathbf{s}^2 + \sqrt{2}\mathbf{s} + 1}. \tag{40}$$

The order of the filter is simply the order of the denominator polynomial in $\mathbf{H}(\mathbf{s})$. Because an n-th order polynomial has n roots, an n-order filter has n poles. The number of poles is equal to the number of "memory elements" (i.e. capacitors or inductors) needed to instantiate the filter. The transfer function (40) is the transfer function for a 2nd-order filter with corner frequency $\omega_c = 1$ rad/s, but the requirement is $\omega_c = 100\pi$ krads/s. This adjustment is made by dividing $\mathbf{s}$ by ω_c:

$$\mathbf{H}(\mathbf{s}) = \frac{1}{\left(\dfrac{\mathbf{s}}{100\pi \times 10^3}\right)^2 + \sqrt{2}\left(\dfrac{\mathbf{s}}{100\pi \times 10^3}\right) + 1}. \tag{41}$$

2. The realization of this filter is based on the Sallen-Key filter topology. The Sallen-Key topology for a second order low-pass filter is shown below. (This is reproduced from Figure 6-42, pg. 285 of the text.)



(This is also the op-amp circuit of Problem 4.32.) The s-domain transfer function is

$$\mathbf{H}(\mathbf{s}) = \frac{1}{C_1C_2R_1R_2\mathbf{s}^2 + (R_1 + R_2)C_2\mathbf{s} + 1}$$

(see Example 6-11, pp. 285–286 of the text). The desired low-pass filter results from choosing the component values (R_1, R_2, C_1, and C_2) to generate the transfer function (41). The equations are

$$\begin{aligned} C_1C_2R_1R_2 &= \frac{1}{(100\pi \times 10^3)^2} \\ (R_1 + R_2)C_2 &= \frac{\sqrt{2}}{100\pi \times 10^3}. \end{aligned} \tag{42}$$

We have two equations in four unknowns. From a purely mathematical point of view, this is an underdetermined system of equations for which there are an infinite number of solutions. But given the fact that resistors and capacitors are only available in a discrete set of resistance and capacitance values, respectively, the problem is not not as unsolvable as it might first appear.

To generalize, the design a continuous-time filter is a three step process. First, the filter requirements must be defined. Second, the filter requirements are used to determine the transfer function

$$\mathbf{H}(\mathbf{s}) = \frac{\mathbf{B}(\mathbf{s})}{\mathbf{A}(\mathbf{s})}.$$

Third, a circuit topology is chosen and the component values are selected to realize the desired transfer function $\mathbf{H}(\mathbf{s})$.

The design of a discrete-time 2nd-order Butterworth lowpass filter proceeds in much the same way. The starting point is the continuos-time filter design in the $\mathbf{s}$-domain. Next, the $\mathbf{s}$-domain transfer function is converted to a $\mathbf{z}$-domain transfer function. The discrete-time filter follows directly from the $\mathbf{z}$-domain transfer function. The procedure is outlined as follows:

1. First, the requirements of the discrete-time filter are defined (i.e., the cut-off frequency $\Omega_c = \omega_c T_s$ is determined).

2. Second, the transfer function of the corresponding continuous-time 2nd-order Butterworth filter is calculated:
$$\mathbf{H}_c(\mathbf{s}) = \frac{1}{\left(\frac{\mathbf{s}}{\omega_c}\right)^2 + \frac{\sqrt{2}}{\omega_c}\mathbf{s} + 1}. \tag{43}$$
Here, we call this filter the *prototype filter*.

3. Next, the prototype filter is converted to a discrete-time filter. This is accomplished by transforming the continuous-time $\mathbf{s}$-domain transfer function $\mathbf{H}_c(\mathbf{s})$ an equivalent discrete-time $\mathbf{z}$-domain transfer function $\mathbf{H}_d(\mathbf{z})$ using the mapping
$$\mathbf{s} = \frac{2}{T_s}\frac{\mathbf{z}-1}{\mathbf{z}+1}. \tag{44}$$
This mapping is called the *bilinear transform*. The result is
$$\begin{aligned}
\mathbf{H}_d(\mathbf{z}) &= \mathbf{H}_c\left(\frac{2}{T_s}\frac{\mathbf{z}-1}{\mathbf{z}+1}\right) \\
&= \frac{\frac{\Omega_c^2}{4+2\sqrt{2}\Omega_c+\Omega_c^2}\left[\mathbf{z}^2+2\mathbf{z}+1\right]}{\mathbf{z}^2 - \frac{8-2\Omega_c^2}{4+2\sqrt{2}\Omega_c+\Omega_c^2}\mathbf{z} + \frac{4-2\sqrt{2}\Omega_c+\Omega_c^2}{4+2\sqrt{2}\Omega_c+\Omega_c^2}} \qquad (45) \\
&= \frac{\frac{\Omega_c^2}{4+2\sqrt{2}\Omega_c+\Omega_c^2}\left[1+2\mathbf{z}^{-1}+\mathbf{z}^{-2}\right]}{1 - \frac{8-2\Omega_c^2}{4+2\sqrt{2}\Omega_c+\Omega_c^2}\mathbf{z}^{-1} + \frac{4-2\sqrt{2}\Omega_c+\Omega_c^2}{4+2\sqrt{2}\Omega_c+\Omega_c^2}\mathbf{z}^{-2}} \qquad (46)
\end{aligned}$$

The last line is of the form

$$\mathbf{H}_d(\mathbf{z}) = \frac{\mathbf{B}(\mathbf{z})}{\mathbf{A}(\mathbf{z})} = \frac{b_0 + b_1\mathbf{z}^{-1} + b_2\mathbf{z}^{-2}}{1 + a_1\mathbf{z}^{-1} + a_2\mathbf{z}^{-2}} \tag{47}$$

and this defines the recursive IIR filter based on the continuous-time Butterworth filter. Note that for discrete-time filters, we prefer the form (46) over the form (45) because signal processors tend to think of LTI systems in terms of the unit delay operator $\mathbf{z}^{-1}$.

4. For the continuous-time filter, the last step involved designing the circuit required to realize the filter. Here, the last step is a VLSI layout involving registers, multipliers, and adders, or computer code to instantiate the recursion defined by $\mathbf{H}_d(\mathbf{z})$.

The procedure for the design of the discrete-time recursive IIR filter paralleled that for the continuous-time filter. The big difference was the use of the bilinear transform to convert $\mathbf{H}_d(\mathbf{s})$ to $\mathbf{H}_d(\mathbf{z})$. The reason the bilinear transform is used is illustrated in Figure 12. Here, it is best to think of the bilinear transform as a mapping: it maps the complex-valued variable $\mathbf{s}$ to the complex-valued variable $\mathbf{z}$. As a consequence of mapping $\mathbf{s}$ to $\mathbf{z}$, it maps regions in the $\mathbf{s}$-plane to regions in the $\mathbf{z}$-plane. The two $\mathbf{s}$-plane regions of interest are the $j\omega$ axis and the open left-half plane. The bilinear transform (44) maps the $j\omega$ axis in the $\mathbf{s}$-plane to the unit circle in the $\mathbf{z}$-plane and all points in the left-half $\mathbf{s}$-plane to points inside the unit circle in the $\mathbf{z}$-plane. The first property of the mapping is desirable because it maps the continuous-time frequency axis in the $\mathbf{s}$-plane to the discrete-time frequency circle in the $\mathbf{z}$-plane. The second property of the mapping is desirable because it maps a causal stable continuous-time system to a causal stable discrete-time system.

As an example, let's design a discrete-time 2nd-order Butterworth filter operating on data sampled at 100 ksamples/s and with a corner frequency of 1 kHz ($\omega_c = 2000\pi$ rads/s). In this case, $\Omega_c = \omega_c T_s = 2000\pi \times 10^{-5} = 2\pi \times 0.01$ rads/sample. The $\mathbf{s}$-domain transfer function of the continuous-time prototype filter and the $\mathbf{z}$-domain transfer function of the desired discrete-time filter are

$$\mathbf{H}_c(\mathbf{s}) = \frac{1}{\left(\frac{\mathbf{s}}{\omega_c}\right)^2 + \frac{\sqrt{2}}{\omega_c}\mathbf{s} + 1} \tag{48}$$

$$\mathbf{H}_d(\mathbf{z}) = 9.4408 \times 10^{-4} \times \frac{1 + 2\mathbf{z}^{-1} + \mathbf{z}^{-2}}{1 - 1.9112\,\mathbf{z}^{-1} + 0.9150\,\mathbf{z}^{-2}}. \tag{49}$$

To write a computer program that applies this filter an input sequence $x[n]$ to produce an output sequence $y[n]$, the time-domain input/output relations corresponding to (49) is needed. The time-

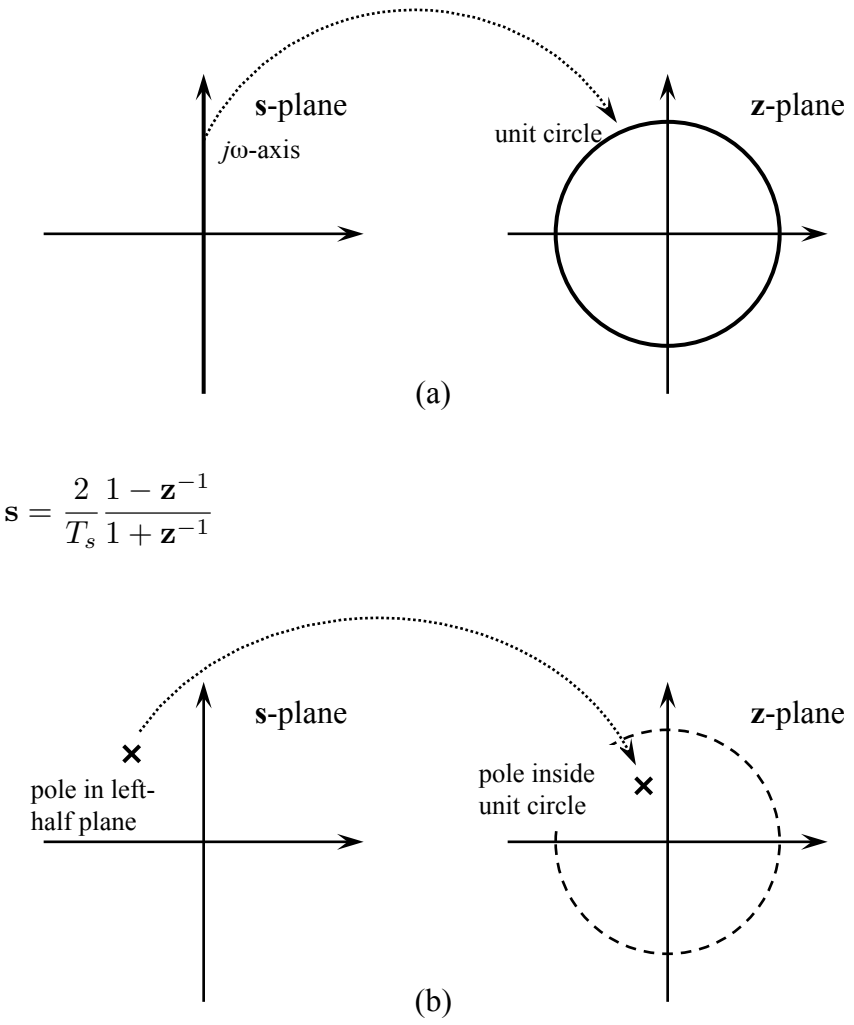


Figure 12: The conformal mapping defined by the bilinear transform (44): (a) the $j\omega$ axis in the $\mathbf{s}$-plane maps to the unit circle in the $\mathbf{z}$-plane; (b) all points in the left-half $\mathbf{s}$-plane map to points inside the unit circle in the $\mathbf{z}$-plane.

domain input/output relationship is

$$y[n] = 1.9112\,y[n-1] - 1.9112\,y[n-2] + \\ 9.4408 \times 10^{-4}x[n] + 18.8816 \times 10^{-4}x[n-1] + 9.4408 \times 10^{-4}x[n-2]. \quad (50)$$

MATLAB® has a built in function that efficiently computes this recursion. The function is `filter`. The following segment of MATLAB® code uses the `filter` command to apply the filter defined by (49) to the input vector `x` to produce the output vector `y`:

```
a = [1 -1.9112 0.9150];          % coefficients of denominator polynomial
b = 9.4408e-4*[1 2 1];           % coefficients of numerator polynomial
y = filter(b,a,x);               % apply the recursive filter to x
                                 % to produce y
```

It is interesting to compare the continuous time prototype filter $\mathbf{H}_c(\mathbf{s})$ with the discrete-time filter $\mathbf{H}_d(\mathbf{z})$. The denominator polynomial of $\mathbf{H}_c(\mathbf{s})$ is

$$\left(\frac{\mathbf{s}}{\omega_c}\right)^2 + \frac{\sqrt{2}}{\omega_c}\mathbf{s} + 1.$$

The roots of this polynomial are

$$-\frac{1}{\sqrt{2}}\omega_c \pm j\frac{1}{\sqrt{2}}\omega_c.$$

These are the poles of $\mathbf{H}_c(\mathbf{s})$. The $\mathbf{s}$-domain pole-zero plot of $\mathbf{H}_c(\mathbf{S})$ is shown in Figure 13. The poles lie on a circle of radius ω_c in the OLHP. (This *is* the Butterworth design criterion. See Section 6-8, pp. 278 – 287 of the text.) The corresponding frequency domain transfer function $\mathbf{H}_c(\omega)$ is also plotted in Figure 13. Here we see the flat passband extending from 0 to about 1000π rads/s. As ω increases, $|\mathbf{H}_c(\omega)|$ decreases a little and crosses -3 dB at $\omega_c = 2000\pi$ rads/s. (This is what it is designed to do.) For the $\omega > \omega_c$, we see the characteristic slope of 40 dB/decade.[1]

The bilinear transform (44) maps the $\mathbf{s}$-domain poles $-4442.9 \pm j4442.9$ to the $\mathbf{z}$-domain poles $0.9556 \pm j0.0425$. The $\mathbf{z}$-domain poles (along with the two $\mathbf{z}$-domain zeros) are plotted in Figure 14. Observe that the poles are located near $\mathbf{z} = e^{j0} = 1$ as expected for a discrete-time lowpass filter. The DTFT $\mathbf{H}_d(e^{j\Omega})$ is $\mathbf{H}_d(\mathbf{z})$ evaluated along the unit circle and is also plotted in Figure 14. Observe that the filter possesses unity gain for $\Omega \approx 0$. For $|\Omega| > 0$ the gain decreases.

[1]The frequency domain transfer function $\hat{\mathbf{H}}(\omega)$ of an n-th order system is characterized by a slope of $20n$ dB/decade for frequencies outside the pass band. See the discussion in Sections 6.1-3 and 6.2-5 of the text, along with the examples described in Sections 6-2.2, 6-2.3, and 6-3.3.

The "40 dB per decade" rule does not apply here. This is because $\mathbf{H}_d(e^{j\Omega})$ is periodic in Ω whereas $\hat{\mathbf{H}}(\omega)$ is not periodic in ω. This is a consequence of the nonlinear relationship between ω and Ω.

To generalize to an n-th order discrete-time Butterworth lowpass filter, one follows the same four steps, except using $\mathbf{D}_n(\mathbf{s})$ in place of $\mathbf{D}_2(\mathbf{s})$ for the continuous-time prototype filter. These steps are repeated here for convenience.

1. First, the requirements of the discrete-time filter are defined (i.e., the cut-off frequency $\Omega_c = \omega_c T_s$ is determined).

2. Second, the transfer function of the corresponding continuous-time n-th order Butterworth filter is calculated:
$$\mathbf{H}_c(\mathbf{s}) = \frac{1}{\mathbf{D}_n(\mathbf{s})}. \tag{51}$$

3. Next, the continuous-time $\mathbf{s}$-domain transfer function $\mathbf{H}_c(\mathbf{s})$ is transformed to an equivalent discrete-time $\mathbf{z}$-domain transfer function $\mathbf{H}_d(\mathbf{z})$ using the bilinear transform (44):
$$\mathbf{H}_d(\mathbf{z}) = \mathbf{H}_c\left(\frac{2}{T_s}\frac{\mathbf{z}-1}{\mathbf{z}+1}\right) = \frac{1}{\mathbf{D}_n\left(\frac{2}{T_s}\frac{\mathbf{z}-1}{\mathbf{z}+1}\right)} \tag{52}$$

4. Filter realization: for a hardware realization, a VLSI layout involving registers, multipliers, and adders completes the design; for a realization in a programmable processor, writing the computer code to perform the recursion corresponding to $\mathbf{H}_d(\mathbf{z})$ completes the design.

Step 3 was hard enough for the second order systems and can be a tedious exercise for $n > 2$. Fortunately MATLAB® has a function that performs this step. Below is a segment of MATLAB® code that designs a 3-rd order discrete-time Butterworth lowpass filter with $\Omega_c = 2\pi \times 0.01$ rads/sample and applies it to a data vector `x`:

```
Wc = 2*pi*0.01;                 % define the cutoff frequency rads/sample
[b,a] = butter(3,Wc/pi);        % use the butter function to produce the
                                % filter coefficients
y = filter(b,a,x);              % apply the recursive filter to x
                                % to produce y
```

The mysterious division by π in the second line is required because of the way the MATLAB® function `butter` normalizes frequency. Typing `help butter` from the MATLAB® prompt gives

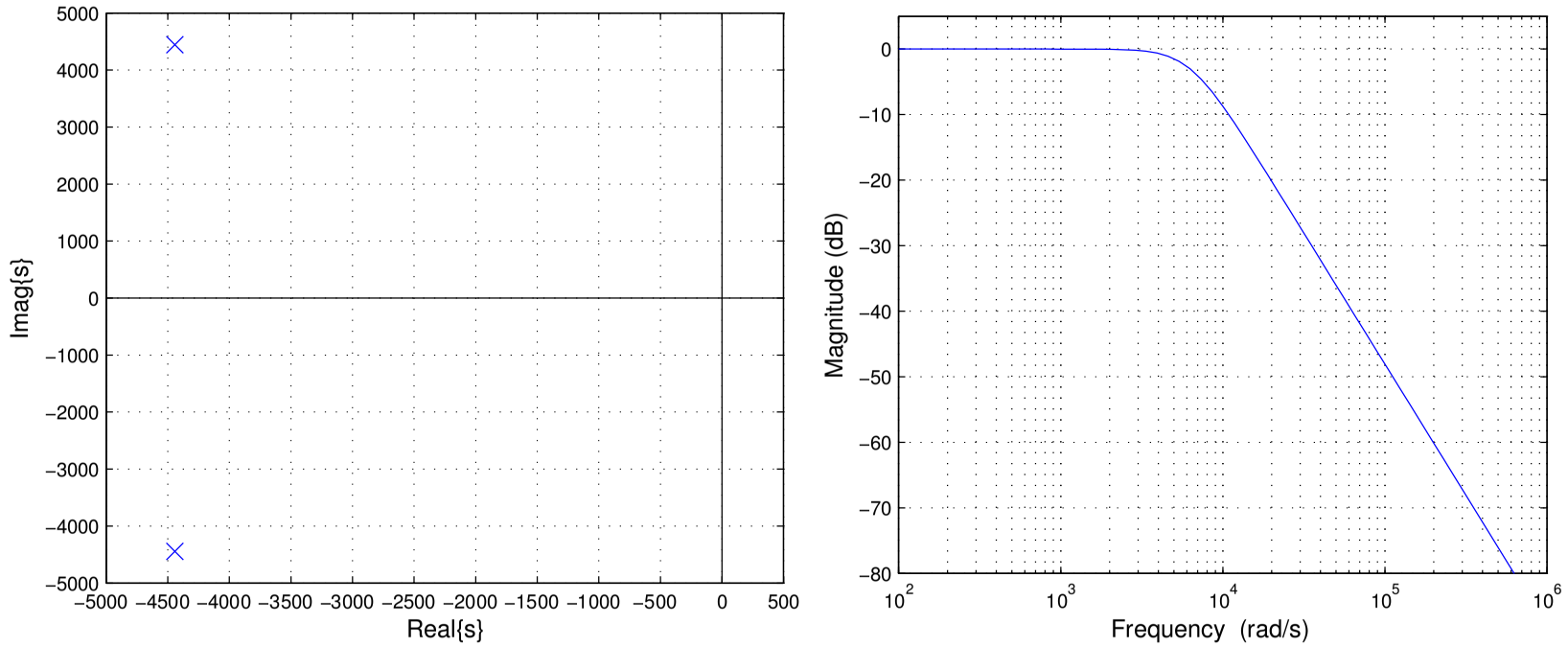


Figure 13: Frequency domain representations of the continuous-time 2nd order Butterworth lowpass filter with transfer function $\mathbf{H}(\mathrm{s})$ given by (41): (left) the s-plane pole-zero plot, (right) the corresponding frequency domain transfer function $\hat{\mathbf{H}}(\omega)$.

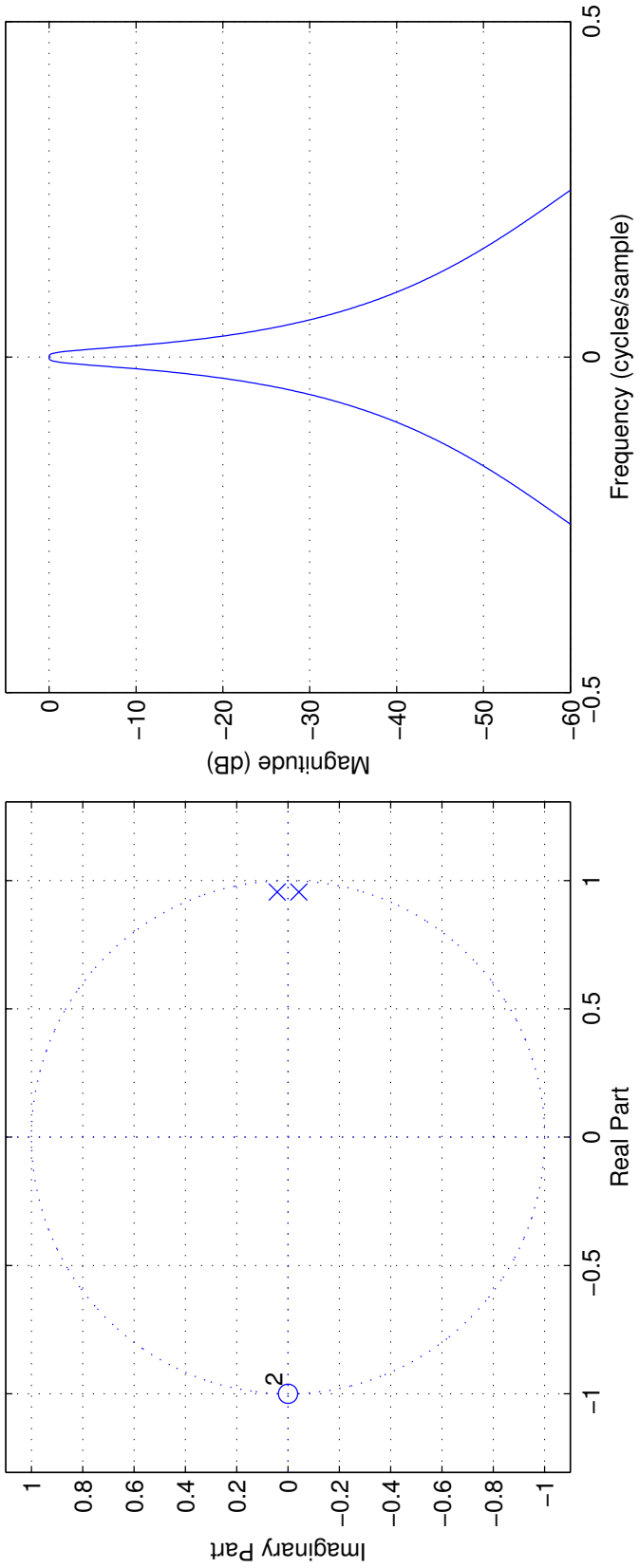


Figure 14: Frequency domain representations of the discrete-time 2nd order Butterworth lowpass filter with transfer function $\mathbf{H}_d(\mathbf{z})$ given by (49): (left) the $\mathbf{z}$-plane pole-zero plot, (right) the corresponding frequency domain transfer function $\mathbf{H}(e^{j\Omega})$.

```
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.
```

This means that the `a` and `b` should be length-4 vectors. The following segment of MATLAB® code shows this to be true:

```
>> [b,a] = butter(3,Wc/pi);
>> b

b =

  1.0e-04 *

  0.2915    0.8744    0.8744    0.2915

>> a

a =

  1.0000   -2.8744    2.7565   -0.8819
```

The DTFT of this recursive filter can be viewed by computing $\mathbf{H}_d(\mathbf{z}) = \mathbf{B}(\mathbf{z})/\mathbf{A}(\mathbf{z})$ at N equally spaced points around the unit circle and plotting the results. The following segment of MATLAB® code produces the plot shown in Figure 15:

```
Wc = 2*pi*0.01;               % define the cutoff frequency rads/sample
[b,a] = butter(3,Wc/pi);      % use the butter function to produce the
                              % filter coefficients
N = 1024;                     % number of points around the unit circle
FF = -0.5:1/N:0.5-1/N;        % the corresponding frequency axis
H = freqz(b,a,N,'whole');     % evaluate transfer function at
                              % N equally spaced points
                              % around the unit circle
figure(1);                    % plot the magnitude of H vs. FF
plot(FF,20*log10(abs(fftshift(H))));
grid on;
xlabel('frequency (cycles/sample)');
ylabel('magnitude (dB)');
axis([-0.5 0.5 -60 3]);
```

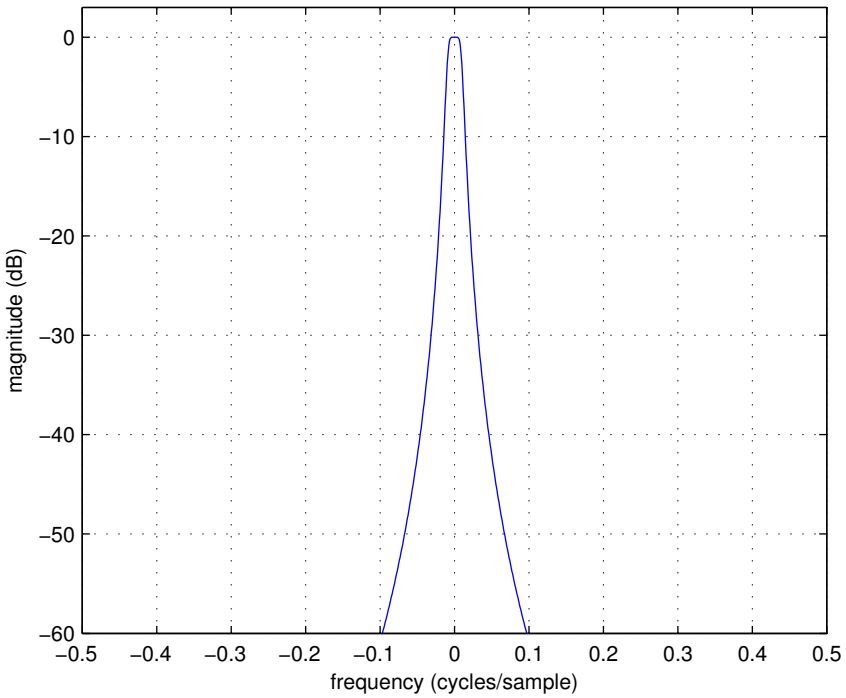


Figure 15: The frequency response of the 3-rd order discrete-time Butterworth filter with $\Omega_c = 2\pi \times 0.01$ rads/sample.