INTRODUCTION TO
Signal Processing

Solutions Manual

Sophocles J. Orfanidis

Department of Electrical & Computer Engineering
Rutgers University, Piscataway, NJ 08855
orfanidi@ece.rutgers.edu

Web page: www.ece.rutgers.edu/~orfanidi/i2sp

Chapter 1 Problems

Problem 1.1

The Nyquist interval is $[-f_s/2, f_s/2] = [-4, 4]$ Hz. The 6 Hz frequency of the wheel lies outside it, therefore, it will be aliased with $f - f_s = 6 - 8 = -2$ Hz. Thus, the wheel will appear to be turning at 2 Hz in the opposite direction.

If $f_s = 12$, the Nyquist interval is $[-6, 6]$. Points on the wheel will appear to be moving up and down, with either positive or negative sense of rotation.

For the other two sampling frequencies, the Nyquist interval is $[-8, 8]$ or $[-12, 12]$ Hz, and therefore, the original 6 Hz frequency lies in it and no aliasing will be perceived.

Problem 1.2

The three terms of the signal correspond to frequencies $f_1 = 1$, $f_2 = 4$, and $f_3 = 6$ Hz. Of these, f_2 and f_3 lie outside the Nyquist interval $[-2.5, 2.5]$. Therefore, they will be aliased with $f_2 - f_s = 4 - 5 = -1$ and $f_3 - f_s = 6 - 5 = 1$ and the aliased signal will be:

$$x_a(t) = 10\sin(2\pi t) + 10\sin(2\pi(-1)t) + 5\sin(2\pi t) = 5\sin(2\pi t)$$

To show that they have the same sample values, we set $t = nT$, with $T = 1/f_s = 1/5$ sec. Then,

$$x(nT) = 10\sin(2\pi n/5) + 10\sin(8\pi n/5) + 5\sin(12\pi n/5)$$

But,

$$\sin(8\pi n/5) = \sin(2\pi n - 2\pi n/5) = -\sin(2\pi n/5)$$

and

$$\sin(12\pi n/5) = \sin(2\pi n + 2\pi n/5) = \sin(2\pi n/5)$$

Thus,

$$\begin{aligned} x(nT) &= 10\sin(2\pi n/5) - 10\sin(2\pi n/5) + 5\sin(2\pi n/5) \\ &= 5\sin(2\pi n/5) = x_a(nT). \end{aligned}$$

If $f_s = 10$ Hz, then the Nyquist interval is $[-5, 5]$ Hz and only f_3 lies outside it. It will be aliased with $f_3 - f_s = 6 - 10 = -4$ resulting in the aliased signal:

$$x_a(t) = 10\sin(2\pi t) + 10\sin(8\pi t) + 5\sin(2\pi(-4)t) = 10\sin(2\pi t) + 5\sin(8\pi t)$$

Problem 1.3

Using the trig identity $2\sin\alpha\sin\beta = \cos(\alpha - \beta) - \cos(\alpha + \beta)$, we find:

$$\begin{aligned} x(t) &= \cos(5\pi t) + 4\sin(2\pi t)\sin(3\pi t) = \cos(5\pi t) + 2[\cos(\pi t) - \cos(5\pi t)] \\ &= 2\cos(\pi t) - \cos(5\pi t) \end{aligned}$$

The frequencies in the signal are $f_1 = 0.5$ and $f_2 = 2.5$ kHz. The Nyquist interval is $[-1.5, 1.5]$ kHz, and f_2 lies outside it. Thus, it will be aliased with $f_{2a} = 2.5 - 3 = -0.5$ giving rise to the signal:

$$x_a(t) = 2\cos(2\pi f_1 t) - \cos(2\pi f_{2a} t) = 2\cos(\pi t) - \cos(-\pi t) = \cos(\pi t)$$

A class of signals aliased with $x(t)$ and $x_a(t)$ is obtained by replacing f_1 and f_2 by their shifted versions: $f_1 + mf_s$, $f_2 + nf_s$ resulting in:

$$x_{mn}(t) = 2\cos(\pi t + 6\pi m t) - \cos(\pi t - 6\pi n t)$$

Problem 1.4

Using a trig identity, we write $x(t) = \cos(8\pi t) + \cos(10\pi t) + \cos(2\pi t)$. The frequencies contained in $x(t)$ are thus, 1 Hz, 4 Hz, and 5 Hz. If the sampling rate is 5 Hz, then the Nyquist interval is $[-2.5, 2.5]$ Hz, and therefore, the 4 Hz and 5 Hz components lie outside it and will be aliased with the frequencies $4 - 5 = -1$ Hz and $5 - 5 = 0$ Hz, etc.

Problem 1.5

Using trig identities we find:

$$x(t) = \sin(6\pi t)\left[1 + 2\cos(4\pi t)\right] = \sin(6\pi t) + \sin(10\pi t) + \sin(2\pi t)$$

with frequency content: $f_1 = 3$, $f_2 = 5$, $f_3 = 1$ kHz. The Nyquist interval is $[-2, 2]$ kHz, and the aliased frequencies are:

$$f_{1a} = f_1 - f_s = 3 - 4 = -1, \quad f_{2a} = f_2 - f_s = 5 - 4 = 1, \quad f_{3a} = f_3 = 1$$

Thus,

$$x_a(t) = \sin(-2\pi t) + \sin(2\pi t) + \sin(2\pi t) = \sin(2\pi t)$$

Problem 1.6

Use the trigonometric identity $2\cos a \cos b = \cos(a + b) + \cos(a - b)$ three times to get

$$\begin{aligned} x(t) &= 2[\cos(10\pi t) + \cos(6\pi t)]\cos(12\pi t) \\ &= \cos(22\pi t) + \cos(2\pi t) + \cos(18\pi t) + \cos(6\pi t) \end{aligned}$$

The frequencies present in this signal are $f_1 = 11$, $f_2 = 1$, $f_3 = 9$, and $f_4 = 3$ Hz. With a sampling rate of 10 Hz, only f_1 and f_3 lie outside the Nyquist interval $[-5, 5]$ Hz, and they will be aliased with $f_1 - f_s = 11 - 10 = 1$ Hz and $f_3 - f_s = 9 - 10 = -1$ Hz. The aliased signal will be:

$$\begin{aligned} x_a(t) &= \cos(2\pi(1)t) + \cos(2\pi t) + \cos(2\pi(-1)t) + \cos(6\pi t) \\ &= 3\cos(2\pi t) + \cos(6\pi t) \end{aligned}$$

To prove the equality of the samples, replace $t = nT = n/10$, because $T = 1/f_s = 1/10$. Then,

$$x(nT) = \cos(22\pi n/10) + \cos(2\pi n/10) + \cos(18\pi n/10) + \cos(6\pi n/10)$$

But, $\cos(22\pi n/10) = \cos(2\pi n + 2\pi n/10) = \cos(2\pi n/10)$ and similarly, $\cos(18\pi n/10) = \cos(2\pi n - 2\pi n/10) = \cos(2\pi n/10)$. Therefore, the sample values become

$$\begin{aligned} x(nT) &= \cos(2\pi n/10) + \cos(2\pi n/10) + \cos(2\pi n/10) + \cos(6\pi n/10) \\ &= 3\cos(2\pi n/10) + \cos(6\pi n/10) = x_a(nT) \end{aligned}$$

If $f_s = 12$ Hz, then f_1 and f_3 will lie outside of $[-6, 6]$ and will be aliased with $11 - 12 = -1$ and $9 - 12 = -3$. The aliased signal will be:

$$\begin{aligned} x_a(t) &= \cos(2\pi(-1)t) + \cos(2\pi t) + \cos(2\pi(-3)t) + \cos(6\pi t) \\ &= 2\cos(2\pi t) + 2\cos(6\pi t) \end{aligned}$$

Problem 1.7

We use the same technique as in the square-wave Example 1.4.6. At a sampling rate of 8 Hz, the signal frequencies of

$$\{1, 3, 5, 7, 9, 11, 13, 15, \ldots\}$$

will be aliased with:

$$\{1, 3, -3, -1, 1, 3, -3, -1, \ldots\}$$

Therefore only $\sin(2\pi t)$ and $\sin(6\pi t)$ terms will appear in the aliased signal. Thus, we write it in the form:

$$x_a(t) = B\sin(2\pi t) + C\sin(6\pi t)$$

To determine B and C, we demand that $x_a(t)$ and $x(t)$ agree at the sampling instants $t = nT = n/8$, because $T = 1/f_s = 1/8$ sec. Therefore, we demand

$$B\sin(2\pi n/8) + C\sin(6\pi n/8) = x(n/8)$$

Setting $n = 1, 2$, we get two equations

$$\begin{aligned} B\sin(2\pi/8) + C\sin(6\pi/8) &= x(1/8) = 0.5 \\ B\sin(4\pi/8) + C\sin(12\pi/8) &= x(2/8) = 1 \end{aligned} \quad \Rightarrow \quad \begin{aligned} B\frac{1}{\sqrt{2}} + C\frac{1}{\sqrt{2}} &= 0.5 \\ B - C &= 1 \end{aligned}$$

The values for $x(1/8)$ and $x(2/8)$ were obtained by inspecting the triangular waveform. Solving for B and C, we find:

$$B = \frac{\sqrt{2} + 2}{4}, \qquad C = \frac{\sqrt{2} - 2}{4}$$

Problem 1.8

For $f_s = 5$ kHz, we have:

$$x_a(t) = \sin(2\pi f_1 t)$$

For $f_s = 10$ kHz, we have:

$$x_a(t) = 2\sin(2\pi f_1 t) + \sin(2\pi f_2 t)$$

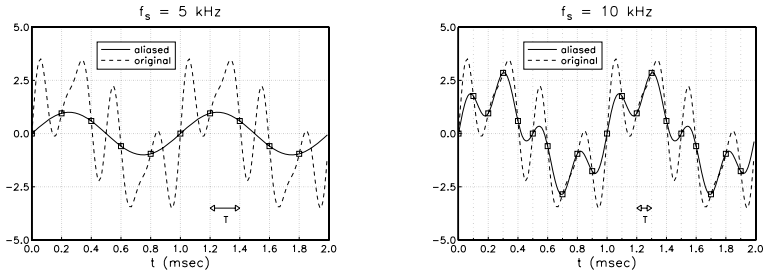


Fig. P1.1 Parts (a,b) of Problem 1.8.

Problem 1.9

The audible and inaudible parts are:

$$x(t) = \underbrace{\sin(10\pi t) + \sin(20\pi t)}_{\text{audible}} + \underbrace{\sin(60\pi t) + \sin(90\pi t)}_{\text{inaudible}}$$

The frequencies of the four terms and their aliased versions are:

$$\begin{aligned} f_A &= 5 & f_{Aa} &= 5 \\ f_B &= 10 & f_{Ba} &= 10 \\ f_C &= 30 \quad \Rightarrow & f_{Ca} &= 30 - 40 = -10 \\ f_D &= 45 & f_{Da} &= 45 - 40 = 5 \end{aligned}$$

a. When there is no prefilter the aliased signal at the output of the reconstructor will be:

$$y_a(t) = \sin(10\pi t) + \sin(20\pi t) + \sin(-20\pi t) + \sin(10\pi t) = 2\sin(10\pi t)$$

b. When there is a perfect prefilter, the f_C, f_D components are removed prior to sampling thus, the sampled and reconstructed signal will remain unchanged, that is, the audible part of $x(t)$:

$$y_a(t) = \sin(10\pi t) + \sin(20\pi t)$$

c. The attenuations introduced by the practical prefilter shown in Fig. P1.2 are obtained by determining the number of octaves to the frequencies f_C, f_D and multiplying by the filter's attenuation of 48 dB/octave:

$$\log_2\left(\frac{f_C}{f_s/2}\right) = \log_2\left(\frac{30}{20}\right) = 0.585 \quad \Rightarrow \quad A_C = 48 \times 0.585 = 28.08 \text{ dB}$$

$$\log_2\left(\frac{f_D}{f_s/2}\right) = \log_2\left(\frac{45}{20}\right) = 1.170 \quad \Rightarrow \quad A_D = 48 \times 1.170 = 56.16 \text{ dB}$$

and in absolute units:

$$|H_C| = 10^{-A_C/20} = 10^{-28.08/20} = 0.0394$$

$$|H_D| = 10^{-A_D/20} = 10^{-56.16/20} = 0.0016$$

Thus, the resulting reconstructed signal will be:

$$\begin{aligned} y_a(t) &= \sin(10\pi t) + \sin(20\pi t) + |H_C|\sin(-20\pi t) + |H_D|\sin(10\pi t) \\ &= 1.0016\sin(10\pi t) + 0.9606\sin(20\pi t) \end{aligned}$$

which agrees closely with the audible part of $x(t)$.

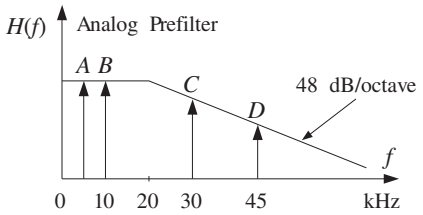


Fig. P1.2 Prefilter specifications of Problem 1.9.

Problem 1.10

The Fourier series expansion of the periodic function $s(t)$ is:

$$s(t) = \sum_{m=-\infty}^{\infty} c_m e^{2\pi jmt/T}, \qquad c_m = \frac{1}{T}\int_{-T/2}^{T/2} s(t)\, e^{-2\pi jmt/T}\, dt$$

But within the basic period $[-T/2, T/2]$, $s(t)$ equals $\delta(t)$. Thus,

$$c_m = \frac{1}{T}\int_{-T/2}^{T/2} \delta(t)\, e^{-2\pi jmt/T}\, dt = \frac{1}{T}$$

Using the delta function integral representation:

$$\int_{-\infty}^{\infty} e^{-2\pi jft}\, dt = \delta(f)$$

we find:

$$\begin{aligned} S(f) &= \int_{-\infty}^{\infty} s(t)\, e^{-2\pi jft}\, dt = \frac{1}{T}\sum_{m=-\infty}^{\infty}\int_{-\infty}^{\infty} s(t)\, e^{-2\pi j(f-mf_s)t}\, dt \\ &= \frac{1}{T}\sum_{m=-\infty}^{\infty} \delta(f - mf_s) \end{aligned}$$

Problem 1.11

This is simply the Fourier series expansion of the periodic function $\hat{X}(f)$ in the variable f, and $x(nT)$ are the Fourier series expansion coefficients calculated by integrating over one period, that is, over $[-f_s/2, f_s/2]$.

Problem 1.12

Write $x(t) = 0.5e^{2\pi j f_0 t} + 0.5e^{-2\pi j f_0 t}$, with Fourier transform:

$$X(f) = 0.5\delta(f - f_0) + 0.5\delta(f + f_0)$$

Its replication gives the spectrum of the sampled signal:

$$\hat{X}(f) = \frac{1}{T}\sum_{m=-\infty}^{\infty} X(f - mf_s) = \frac{1}{2T}\sum_{m=-\infty}^{\infty} [\delta(f - mf_s - f_0) + \delta(f - mf_s + f_0)]$$

$$= \frac{1}{2T}\sum_{m=-\infty}^{\infty} [\delta(f - mf_s - f_0) + \delta(f + mf_s + f_0)]$$

where in the second term, we changed the summation index from m to $-m$.

Problem 1.13

We have for the complex sinusoid, the Laplace, Fourier, and magnitude spectra:

$$x(t) = e^{-at}e^{2\pi j f_0 t} = e^{-(a - 2\pi j f_0)t}, \qquad t \geq 0$$

$$X(s) = \frac{1}{s + a - 2\pi j f_0}$$

$$X(f) = \frac{1}{a + 2\pi j(f - f_0)} \quad \Rightarrow \quad |X(f)|^2 = \frac{1}{a^2 + 4\pi^2(f - f_0)^2}$$

The length-L sampled signal $x(nT) = e^{-(a - 2\pi j f_0)nT}$, $n = 0, 1, \ldots, L-1$ has spectrum:

$$\hat{X}_L(f) = \sum_{n=0}^{L-1} x(nT)e^{-2\pi j f nT} = \sum_{n=0}^{L-1} e^{-(a + 2\pi j(f - f_0))nT}$$

$$= \frac{1 - e^{-(a + 2\pi j(f - f_0))LT}}{1 - e^{-(a + 2\pi j(f - f_0))T}}$$

with magnitude spectrum:

$$|\hat{X}_L(f)|^2 = \frac{1 - 2e^{-aLT}\cos(2\pi(f - f_0)LT) + e^{-2aLT}}{1 - 2e^{-aT}\cos(2\pi(f - f_0)T) + e^{-2aT}}$$

In the limit $L \to \infty$, we have $e^{-aLT} \to 0$ and obtain the sampled spectrum of the infinitely long signal:

$$\hat{X}(f) = \sum_{n=0}^{\infty} x(nT)e^{-2\pi j f nT} = \frac{1}{1 - e^{-(a + 2\pi j(f - f_0))T}}$$

with magnitude spectrum:

$$|\hat{X}(f)|^2 = \frac{1}{1 - 2e^{-aT}\cos(2\pi(f - f_0)T) + e^{-2aT}}$$

The continuous time limit $T \to 0$ of $\hat{X}(f)$ gives rise to the analog spectrum:

$$\lim_{T\to 0} T\hat{X}(f) = \lim_{T\to 0} \frac{T}{1 - e^{-(a + 2\pi j(f - f_0))T}} = \frac{1}{a + 2\pi j(f - f_0)} = X(f)$$

Fig. P1.3 shows the required spectra. From the left graph, we see that as f_s increases, the sampled spectrum agrees more closely with the analog spectrum within the Nyquist interval. From the right graph, we see that the spectrum of the truncated sampled signal has ripples that follow the general shape of $\hat{X}(f)$. As $L \to \infty$, the sampled spectrum $\hat{X}_L(f)$ will tend to $\hat{X}(f)$, which is only an approximation of the true spectrum even within the Nyquist interval.

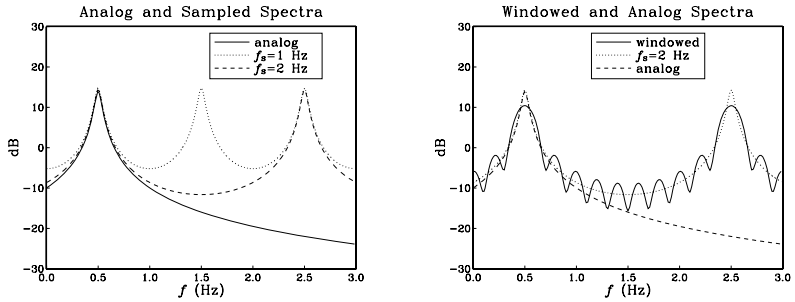


Fig. P1.3 Spectra of analog, sampled, and windowed signals of Problem 1.13.

In the case of real-valued signals, we will also get the spectral peaks at the negative frequency $-f_0$. If f_0 is small, the peaks at $\pm f_0$ will influence each other somewhat, but apart from that the same conclusions can be drawn about the sampled signals.

Problem 1.14

This is the same as Example 1.5.4. In units of dB per decade, the attenuations due to the signal and filter combine to give:

$$A(f) = \alpha \log_{10}\left(\frac{f}{f_{\max}}\right) + \beta \log_{10}\left(\frac{f}{f_{\max}}\right) = \gamma \log_{10}\left(\frac{f}{f_{\max}}\right)$$

where $\gamma = \alpha + \beta$. The requirement $A(f_s - f_{\max}) \geq A$ gives:

$$\gamma \log_{10}\left(\frac{f_s - f_{\max}}{f_{\max}}\right) \geq A \quad \Rightarrow \quad f_s \geq f_{\max} + f_{\max}10^{A/\gamma}$$

Problem 1.15

The first replica of $X_{\text{in}}(f)$, shifted to f_s, will have spectrum:

$$|X_{\text{in}}(f - f_s)| = \frac{1}{\sqrt{1 + (0.1(f - f_s))^8}}$$

Its value at the passband frequency $f_{\text{pass}} = f_{\max} = 20$ kHz will be:

$$|X_{\rm in}(f_{\rm max}-f_s)| = \frac{1}{\sqrt{1+\left(0.1\,(f_{\rm max}-f_s)\right)^8}}$$

This must be suppressed by $A = 60$ dB relative to $|X_{\rm in}(f_{\rm max})|$, thus, we require:

$$-10\log_{10}\left|\frac{X_{\rm in}(f_{\rm max}-f_s)}{X_{\rm in}(f_{\rm max})}\right|^2 = A \quad\Rightarrow\quad 10\log_{10}\left[\frac{1+\left(0.1\,(f_{\rm max}-f_s)\right)^8}{1+(0.1f_{\rm max})^8}\right] = 60$$

which can be solved for f_s to give $f_s = 132.52$ kHz. Using the prefilter, we require that its passband attenuation be $A_{\rm pass} = 1$ dB at $f_{\rm pass} = f_{\rm max} = 20$ kHz, that is,

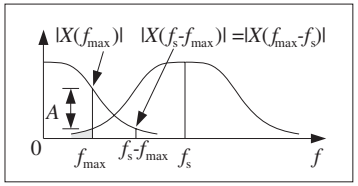
$$-10\log_{10}|H(f_{\rm max})|^2 = 10\log_{10}\left[1+(f_{\rm max}/f_0)^6\right] = A_{\rm pass}$$

which can be solved for f_0 to give $f_0 = 25.05$ kHz. The prefiltered signal that gets sampled is now

$$|X(f)| = |H(f)X_{\rm in}(f)| = \frac{1}{\sqrt{1+(f/f_0)^6}}\,\frac{1}{\sqrt{1+(0.1f)^8}}$$

Its first replica $|X(f-f_s)|$ is required to be suppressed by $A = 60$ dB at $f_{\rm max}$

$$-10\log_{10}\left|\frac{X(f_{\rm max}-f_s)}{X(f_{\rm max})}\right|^2 = A,$$



which gives the condition

$$10\log_{10}\left[\frac{1+\left((f_{\rm max}-f_s)/f_0\right)^6}{1+(f_{\rm max}/f_0)^6}\cdot\frac{1+\left(0.1\,(f_{\rm max}-f_s)\right)^8}{1+(0.1f_{\rm max})^8}\right] = A \qquad \text{(P1.1)}$$

It can be solved approximately by ignoring the +1 in the numerators, that is,

$$10\log_{10}\left[\frac{\left((f_{\rm max}-f_s)/f_0\right)^6}{1+(f_{\rm max}/f_0)^6}\cdot\frac{\left(0.1\,(f_{\rm max}-f_s)\right)^8}{1+(0.1f_{\rm max})^8}\right] = 60\ \text{dB} \qquad \text{(P1.2)}$$

which gives $f_s = 80.1$ kHz. The exact solution of Eq. (P1.1) is $f_s = 80.0633$ kHz. The first factor in Eq. (P1.2) represents the stopband attenuation due to the filter, and the second the attenuation due to the signal. With $f_s = 80.1$ kHz, we have

$$10\log_{10}\left[\frac{1+\left((f_{\rm max}-f_s)/f_0\right)^6}{1+(f_{\rm max}/f_0)^6}\right] = A_{\rm stop} = 21.8\ \text{dB}$$

$$10\log_{10}\left[\frac{1+\left(0.1\,(f_{\rm max}-f_s)\right)^8}{1+(0.1f_{\rm max})^8}\right] = A - A_{\rm stop} = 38.2\ \text{dB}$$

Thus, the use of the prefilter enables us to reduce the sampling rate from 132 kHz to 80 kHz. Note also, that for *large* f, the spectrum $|X(f)|$ follows the power law

$$|X(f)|^2 = \frac{1}{1+(f/f_0)^6}\cdot\frac{1}{1+(0.1f)^8} \approx \frac{\text{const.}}{f^6}\cdot\frac{\text{const.}}{f^8} = \frac{\text{const.}}{f^{14}}$$

giving rise to an effective attenuation

$$-10\log_{10}|X(f)|^2 = \gamma\log_{10}f + \text{const.}$$

where $\gamma = 140$ dB/decade, of which 60 dB/decade is due to the filter and 80 dB/decade due to the signal. The approximate method of Example 1.5.4 or Problem 1.14 would have given here $f_s = f_{\rm max} + 10^{A/\gamma}f_{\rm max} = 20 + 20\cdot 10^{60/140} = 73.65$ kHz, which is fairly close to the more accurate value of 80 kHz.

Problem 1.16

The passband condition at $f = f_{\rm max} = 20$ kHz is:

$$10\log_{10}\left[1+(f_{\rm max}/f_0)^{2N}\right] = A_{\rm pass} \qquad \text{(P1.3)}$$

which can be solved for f_0 in terms of N as follows:

$$f_0^{2N} = \frac{f_{\rm max}^{2N}}{r-1}, \qquad \text{where } r = 10^{A_{\rm pass}/10}$$

The antialiasing condition, replacing Eq. (P1.1), is now

$$10\log_{10}\left[\frac{1+\left((f_{\rm max}-f_s)/f_0\right)^{2N}}{1+(f_{\rm max}/f_0)^{2N}}\cdot\frac{1+\left(0.1\,(f_{\rm max}-f_s)\right)^8}{1+(0.1f_{\rm max})^8}\right] = A$$

or,

$$\frac{1+\left((f_{\rm max}-f_s)/f_0\right)^{2N}}{1+(f_{\rm max}/f_0)^{2N}}\cdot\frac{1+\left(0.1\,(f_{\rm max}-f_s)\right)^8}{1+(0.1f_{\rm max})^8} = 10^{A/10}$$

Replacing f_0^{2N} by its expression above, we obtain

$$\frac{1+\left((f_{\rm max}-f_s)/f_{\rm max}\right)^{2N}(r-1)}{r}\cdot\frac{1+\left(0.1\,(f_{\rm max}-f_s)\right)^8}{1+(0.1f_{\rm max})^8} = 10^{A/10}$$

and solving for N, we have

$$N = \frac{\ln\left(\dfrac{10^{A/10}rR(f_s)-1}{r-1}\right)}{2\ln\left(\dfrac{f_s-f_{\rm max}}{f_{\rm max}}\right)} \qquad \text{(P1.4)}$$

where $R(f_s)$ is defined by

$$R(f_s) = \frac{1+(0.1f_{\rm max})^8}{1+\left(0.1\,(f_{\rm max}-f_s)\right)^8}$$

Eq. (P1.4) gives the minimum prefilter order for a desired suppression level A and rate f_s. With $f_s = 70$ kHz, $A = 60$ dB, $A_{\rm pass} = 1$ dB, we find

$$r = 10^{A_{\rm pass}/10} = 10^{0.1} = 1.259, \quad 10^{A/10}R(f_s) = 657.918$$

and

$$N = \frac{\ln(3194.281)}{2\ln(50/20)} = 4.403 \quad\Rightarrow\quad N = 5$$

We calculate also $f_0 = f_{\rm max}/(r-1)^{1/2N} = 22.9$ kHz.

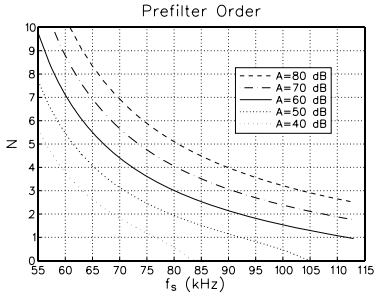


Fig. P1.4 Butterworth order versus sampling rate.

Problem 1.17

Figure P1.4 shows a plot of N given by Eq. (P1.4) as a function of f_s for different choices of A. Notice the two data points on the $A = 60$ curve at $f_s = 70$ and $f_s = 80$ kHz that were computed in this and the last example. It is evident from this graph that the complexity of the prefilter increases with A, at any fixed f_s; and that it decreases with f_s, for fixed A.

Problem 1.18

Using the results of Problem 1.14, we solve for the combined attenuation γ of signal and prefilter:

$$A = \gamma \log_{10}\left(\frac{f_s - f_{\max}}{f_{\max}}\right) \quad \Rightarrow$$

where $A = 60$ dB, $f_s = 30$ kHz and $f_{\max} = 10$ kHz. We find:

$$\gamma = \frac{A}{\log_{10}\left(\dfrac{f}{f_{\max}}\right)} = \frac{60}{\log_{10}\left(\dfrac{30-10}{10}\right)} = 200$$

Because the signal has $\alpha = 80$ dB/decade, we find the filter's attenuation:

$$\beta = \gamma - \alpha = 200 - 80 = 120 \text{ dB/decade}$$

which translates into a filter order:

$$\beta = 20N \quad \Rightarrow \quad N = \beta/20 = 120/20 = 6$$

If we increase the sampling rate to 50 kHz, the combined attenuation will be $\gamma = 100$, which gives $\beta = 100 - 80 = 20$ dB/decade and order $N = \beta/20 = 1$.

Problem 1.19

This problem is the generalization of Prob. 1.15. If we do not use a prefilter, the requirement that the aliased components (of the 1st replica) be suppressed by A dB relative to the signal components can be expressed in the form:

$$\frac{|X_{\rm in}(f_s - f_{\max})|^2}{|X_{\rm in}(f_{\max})|^2} = 10^{-A/10}$$

Inserting the given expression for $|X_{\rm in}(f)|$, we get the desired equation for f_s in terms of A and $f_{\max}$:

$$\frac{1 + (f_{\max}/f_a)^{2Na}}{1 + \left((f_s - f_{\max})/f_a\right)^{2Na}} = 10^{-A/10}$$

If we use a Butterworth prefilter of order N:

$$|H(f)|^2 = \frac{1}{1 + (f/f_0)^{2N}}$$

then, the 3-dB frequency f_0 is determined by requiring that at the edge of the passband the attenuation be B dB:

$$|H(f_{\max})|^2 = \frac{1}{1 + (f_{\max}/f_0)^{2N}} = 10^{-B/10}$$

The filtered input that gets sampled will now have spectrum $X(f) = H(f)X_{\rm in}(f)$, which gives:

$$|X(f)|^2 = |H(f)|^2 |X_{\rm in}(f)|^2 = \frac{1}{1 + (f/f_0)^{2N}} \cdot \frac{1}{1 + (f/f_a)^{2Na}}$$

The antialiasing condition $|X(f_s - f_{\max})|^2 = 10^{-A/10}|X(f)|^2$ gives now:

$$\frac{1}{1 + \left((f_s - f_{\max})/f_0\right)^{2N}} \cdot \frac{1}{1 + \left((f_s - f_{\max})/f_a\right)^{2Na}} = 10^{-A/10} \cdot \frac{1}{1 + (f_{\max}/f_0)^{2N}} \cdot \frac{1}{1 + (f_{\max}/f_a)^{2Na}}$$

Using the asymptotic forms of the spectra, valid for large f's, that is,

$$|X_{\rm in}(f)|^2 \simeq \frac{1}{(f/f_a)^{2Na}}, \qquad |H(f)|^2 \simeq \frac{1}{(f/f_0)^{2N}}$$

the antialiasing condition would read:

$$\frac{(f_s - f_{\max})^{2N}}{f_0^{2N}} \cdot \frac{(f_s - f_{\max})^{2Na}}{f_a^{2Na}} = 10^{-A/10} \frac{f_{\max}^{2N}}{f_0^{2N}} \cdot \frac{f_{\max}^{2Na}}{f_a^{2Na}}$$

which simplifies into:

$$\left(\frac{f_s - f_{\max}}{f_{\max}}\right)^{2(N+Na)} = 10^{A/10}$$

or,

$$f_s = f_{\max} + f_{\max} 10^{A/20(N+Na)}$$

It agrees with the results of Problem 1.14, with $\gamma = \alpha + \beta = 20Na + 20N$. For any fixed desired value of A, the limit $N \to \infty$ gives

$$f_s = f_{\max} + f_{\max} 10^{A/20(N+Na)} \to f_{\max} + f_{\max} = 2f_{\max}$$

In this case, the Butterworth filter resembles more and more a brick-wall filter with cutoff $f_{\max}$.

Problem 1.20

The linear relationship between N and A follows from the approximation of Problem 1.19:

$$\left(\frac{f_s - f_{\max}}{f_{\max}}\right)^{2(N+N_a)} = 10^{A/10}$$

or,

$$2(N+N_a)\log_{10}\left(\frac{f_s - f_{\max}}{f_{\max}}\right) = \frac{A}{10}$$

Problem 1.21

The presence of $P(f)$ is called the *aperture* effect. The replicas of $X(f)$ embedded in $\hat{X}(f)$ will still be present, but they will be weighted by $P(f)$. The need for using antialiasing prefilters to avoid overlapping of replicas remains the same. For a pulse of width τ, we have:

$$|P(f)| = \tau \left|\frac{\sin(\pi f \tau)}{\pi f \tau}\right|$$

Therefore, the narrower the width τ the flatter the spectrum $P(f)$. If $\tau \ll T$, then $P(f)$ will be essentially *flat* within the Nyquist interval, thus, leaving the central replica almost unchanged. Fig. P1.5 shows $P(f)$ relative to the replicas of $\hat{X}(f)$ for the case $\tau = T/5$.

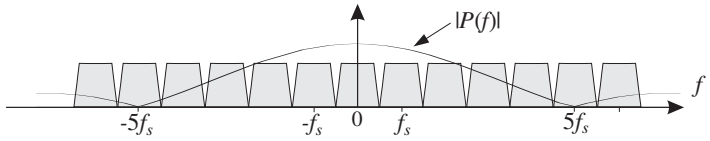


Fig. P1.5 Butterworth order versus sampling rate.

Problem 1.22

The stopband frequency of the postfilter is: $f_{\text{stop}} = f_s - f_c = 6 - 1 = 5$ kHz. At this frequency, the attenuation of the staircase DAC is

$$A_{\text{DAC}} = -20\log_{10}\left|\frac{\sin(\pi f_{\text{stop}}/f_s)}{\pi f_{\text{stop}}/f_s}\right| = 14.4 \text{ dB}$$

Thus, the stopband attenuation of the postfilter at $f = f_{\text{stop}}$ must be:

$$A_{\text{stop}} = A - A_{\text{DAC}} = 40 - 14.4 = 25.6 \text{ dB}$$

Problem 1.23

Using convolution in the time domain, we get:

$$x_a(t) = \int_{-\infty}^{\infty} h(t-t')\hat{x}(t')\,dt' = \sum_n x(nT)\int_{-\infty}^{\infty} h(t-t')\,\delta(t'-nT)\,dt'$$

$$= \sum_n x(nT)h(t-nT)$$

Working in the frequency domain we have:

$$X_a(f) = H(f)\hat{X}(f) = \sum_n x(nT)\left[e^{-2\pi jfnT}H(f)\right]$$

Using the delay theorem of Fourier transforms, we recognize the term in the bracket as the FT of $h(t-nT)$. Thus, going to the time domain, we get:

$$x_a(t) = \sum_n x(nT)h(t-nT)$$

Problem 1.24

Start with a complex sinusoid $x(t) = e^{2\pi jf_0 t}$. Its sampled spectrum will be:

$$\hat{X}(f) = \frac{1}{T}\sum_{m=-\infty}^{\infty} X(f-mf_s) = \frac{1}{T}\sum_{m=-\infty}^{\infty}\delta(f-f_0-mf_s)$$

The spectrum of the reconstructed analog signal will be then:

$$X_{\text{rec}}(f) = H(f)\hat{X}(f) = \frac{1}{T}\sum_{m=-\infty}^{\infty} H(f)\,\delta(f-f_0-mf_s) = \frac{1}{T}\sum_{m=-\infty}^{\infty} H(f_0+mf_s)\,\delta(f-f_0-mf_s)$$

Taking inverse Fourier transforms, we find:

$$x_{\text{rec}}(t) = \frac{1}{T}\sum_{m=-\infty}^{\infty} H(f_0+mf_s)\,e^{2\pi j(f_0+mf_s)t}$$

Thus, $f_m = f_0 + mf_s$. Writing

$$\frac{1}{T}H(f_m) = A_m e^{j\theta_m}, \quad \text{so that} \quad A_m = \frac{1}{T}|H(f_m)|, \quad \theta_m = \arg H(f_m)$$

we have:

$$x_{\text{rec}}(t) = \sum_{m=-\infty}^{\infty} A_m e^{2\pi j(f_0+mf_s)t + j\theta_m}$$

Taking imaginary parts, we find for the real sinusoidal case:

$$x_{\text{rec}}(t) = \sum_{m=-\infty}^{\infty} A_m \sin\big(2\pi(f_0+mf_s)t + \theta_m\big)$$

Assuming $|f_0| < f_s/2$, for the staircase reconstructor we have:

$$\frac{1}{T}H(f_m) = \frac{\sin(\pi f_m T)}{\pi f_m T}e^{-j\pi f_m T}$$

and because $\pi f_m T = \pi f_0 T + m\pi$, we have:

$$\frac{1}{T}H(f_m) = \frac{\sin(\pi f_0 T)}{\pi f_m T}e^{-j\pi f_0 T}$$

Thus,

$$A_m = \frac{\sin(\pi f_0 T)}{\pi f_m T}, \qquad \theta_m = -\pi f_0 T$$

For the ideal reconstructor, we have $\theta_m = 0$ and $A_m = 0$ except at $m = 0$ where $A_0 = 1$, that is, $A_m = \delta_m$. Thus,

$$x_{\text{rec}}(t) = \sum_{m=-\infty}^{\infty}\delta_m \sin\big(2\pi(f_0+mf_s)t\big) = \sin(2\pi f_0 t)$$

Problem 1.25

Take the linear combination of the results of the single sinusoid case of Eq. (1.6.12).

Chapter 2 Problems

Problem 2.1

The quantized values are:

analog value	quantized value	DAC output
2.9	2	0 0 1
3.1	4	0 1 0
3.7	4	0 1 0
4	4	0 1 0
-2.9	-2	1 1 1
-3.1	-4	1 1 0
-3.7	-4	1 1 0
-4	-4	1 1 0

For the offset binary case, complement the first bit in this table.

Problem 2.2

Filling the table, we have

n	$x(n)$	$x_Q(n)$	offset binary	2's complement
0	0.000	0.000	1 0 0 0	0 0 0 0
1	1.243	1.000	1 0 0 1	0 0 0 1
2	2.409	2.000	1 0 1 0	0 0 1 0
3	3.423	3.000	1 0 1 1	0 0 1 1
4	4.222	4.000	1 1 0 0	0 1 0 0
5	4.755	5.000	1 1 0 1	0 1 0 1
6	4.990	5.000	1 1 0 1	0 1 0 1
7	4.911	5.000	1 1 0 1	0 1 0 1
8	4.524	5.000	1 1 0 1	0 1 0 1
9	3.853	4.000	1 1 0 0	0 1 0 0
10	2.939	3.000	1 0 1 1	0 0 1 1
11	1.841	2.000	1 0 1 0	0 0 1 0
12	0.627	1.000	1 0 0 1	0 0 0 1
13	-0.627	-1.000	0 1 1 1	1 1 1 1
14	-1.841	-2.000	0 1 1 0	1 1 1 0
15	-2.939	-3.000	0 1 0 1	1 1 0 1
16	-3.853	-4.000	0 1 0 0	1 1 0 0
17	-4.524	-5.000	0 0 1 1	1 0 1 1
18	-4.911	-5.000	0 0 1 1	1 0 1 1
19	-4.990	-5.000	0 0 1 1	1 0 1 1

Problem 2.3

With $R = 10$ volts and $e_{\text{rms}} \leq 10^{-3}$ volts, we have using $Q = \sqrt{12}e_{\text{rms}}$:

$$2^B = \frac{R}{Q} = \frac{R}{\sqrt{12}e_{\text{rms}}} \geq \frac{10}{\sqrt{12} \cdot 10^{-3}}$$

which gives

$$B \geq \log_2 \left[\frac{10}{\sqrt{12} \cdot 10^{-3}} \right] = 11.495$$

Therefore, $B = 12$. The actual rms error will be

$$e_{\text{rms}} = \frac{R}{2^B \sqrt{12}} = \frac{10}{2^{12}\sqrt{12}} = 0.705 \text{ mvolt.}$$

The dynamic range will be $6B = 6 \cdot 12 = 72$ dB.

Problem 2.4

At 44.1 kHz sampling rate, one minute of music has 60 sec containing 60×44100 samples. And, for two stereo channels, $2 \times 60 \times 44100$ samples. If each sample is quantized with 16 bits, the total number of bits will be

$$2 \times 60 \times 44100 \times 16 \text{ bits}$$

or, dividing by 8 we get the total number of bytes:

$$\frac{2 \times 60 \times 44100 \times 16}{8} \text{ bytes}$$

Dividing by 1024, we get kbytes and by another 1024, Mbytes:

$$\frac{2 \times 60 \times 44100 \times 16}{8 \times 1024 \times 1024} = 10.09 \text{ Mbytes}$$

Problem 2.5

Following similar steps as in the previous problem, we have for part (a):

$$\frac{(3 \times 60 \text{ sec})\,(16 \text{ channels})\,(48000 \text{ samples/sec})\,(20 \text{ bits/sample})}{(8 \text{ bits/byte})\,(1024^2 \text{ bytes/Mbyte})} = 330 \text{ Mbytes}$$

that is, practically the entire hard disk of a typical home PC. For part (b), if in each channel we must perform $N_{\text{MAC}} = 35$ operations on each sample, then these operations must be finished during the sampling interval $T = 1/f_s$ between samples:

$$N_{\text{MAC}} T_{\text{MAC}} \leq T \qquad \Rightarrow \qquad T_{\text{MAC}} \leq \frac{T}{N_{\text{MAC}}} = \frac{1}{N_{\text{MAC}} f_s}$$

which gives for the time per MAC:

$$T_{\text{MAC}} = \frac{1}{35 \cdot 48 \text{ kHz}} = 0.595 \cdot 10^{-3} \text{ msec} = 595 \text{ nsec}$$

This is plenty of time for today's DSP chips, which can perform a MAC operation within 30–80 nsec. If a single DSP chip handles all 16 channels, then the total number of MACs that must be performed within the sampling interval T are $16N_{\text{MAC}}$. This gives:

$$T_{\text{MAC}} = \frac{1}{16 \cdot 35 \cdot 48 \text{ kHz}} = \frac{595}{16} = 37 \text{ nsec}$$

which is just within the capability of the fastest DSP chips.
In practice, a typical professional digital recording system would require the processing and mixing of at least 32 channels. The above results suggest that one use at least two DSPs to multiplex the required operations.

Problem 2.6

The mean is

$$m_e = \int_{-Q}^{0} e p(e)\, de = \frac{1}{Q} \int_{-Q}^{0} e\, de = -\frac{Q}{2}$$

and the variance:

$$\sigma_e^2 = E[(e - m_e)^2] = \int_{-Q}^{0} (e - m_e)^2 p(e)\, de = \frac{1}{Q} \int_{-Q}^{0} \left(e + \frac{Q}{2}\right)^2 de = \frac{Q^2}{12}$$

Problem 2.7

Solving Eq. (2.2.10) for L in terms of ΔB, we find:

$$L = \left[\frac{2^{2\Delta B} \pi^{2p}}{2p+1} \right]^{\frac{1}{2p+1}} \tag{P2.1}$$

With $\Delta B = 15$ and $p = 1, 2, 3$, we find:

$$L = 1523,\ 116,\ 40$$

The oversampled audio rates will be, with $f_s = 44.1$ kHz:

$$Lf_s = 67.164,\ 5.114,\ 1.764 \text{ MHz}$$

Problem 2.8

Using Eq. (P2.1) with $\Delta B = 7$ bits and $p = 1, 2, 3$, we find:

$$L = 38,\ 13,\ 8$$

The oversampled speech rates will be, with $f_s = 8$ kHz:

$$Lf_s = 304,\ 104,\ 48 \text{ kHz}$$

Problem 2.9

Replacing $\overline{b}_1 = 1 - b_1$, we have:

$$\begin{aligned} x_Q &= R(\overline{b}_1 2^{-1} + b_2 2^{-2} + \cdots + b_B 2^{-B} - 0.5) \\ &= R((1 - b_1) 2^{-1} + b_2 2^{-2} + \cdots + b_B 2^{-B} - 0.5) \\ &= R(-b_1 2^{-1} + b_2 2^{-2} + \cdots + b_B 2^{-B}) \end{aligned}$$

where the first 2^{-1} canceled the last -0.5.

Problem 2.10

As an example, consider the polynomial of degree $M = 3$: $B(z) = b_0 + b_1 z + b_2 z^2 + b_3 z^3$. Iterating the algorithm for $i = 3, 2, 1, 0$, we find:

$$\begin{aligned} &p = 0 \\ &p = ap + b_3 = b_3 \\ &p = ap + b_2 = ab_3 + b_2 \\ &p = ap + b_1 = a(ab_3 + b_2) + b_1 = a^2 b_3 + ab_2 + b_1 \\ &p = ap + b_0 = a(a^2 b_3 + ab_2 + b_1) + b_0 = a^3 b_3 + a^2 b_2 + ab_1 + b_0 = B(a) \end{aligned}$$

Problem 2.11

```
/* pol.c - polynomial evaluator */

double pol(M, b, a)
int M;
double *b, a;

        int i;
        double p = 0;

        for (i=M; i>=0; i--)
                p = a * p + b[i];

        return p;
```

Problem 2.12

Consider the division of $B(z)$ by $(z-a)$:

$$B(z) = (z-a)Q(z) + R$$

where the quotient polynomial $Q(z)$ must have degree $M-1$, and the remainder R must be a constant. Indeed, setting $z = a$, gives $R = b(a)$. So the objective of the algorithm is to compute R iteratively. Inserting the polynomial expressions we have:

$$b_M z^M + b_{M-1} z^{M-1} + \cdots + b_1 z + b_0 = (z-a)\,(q_{M-1} z^{M-1} + q_{M-2} z^{M-2} + \cdots + q_1 z + q_0) + R$$

Equating like powers of z, we find:

$$\begin{aligned} b_M &= q_{M-1} & q_{M-1} &= b_M \\ b_{M-1} &= q_{M-2} - a q_{M-1} & q_{M-2} &= a q_{M-1} + b_{M-1} \\ b_{M-2} &= q_{M-3} - a q_{M-2} & q_{M-3} &= a q_{M-2} + b_{M-2} \\ &\cdots \quad \Rightarrow & &\cdots \\ b_1 &= q_0 - a q_1 & q_0 &= a q_1 + b_1 \\ b_0 &= R - a q_0 & R &= a q_0 + b_0 \end{aligned}$$

Problem 2.13

For a polynomial of degree $M = 3$, $B(z) = b_1 z + b_2 z^2 + b_3 z^3$, we iterate the algorithm for $i = 3, 2, 1$:

$$\begin{aligned} &p = 0 \\ &p = a(p + b_3) = a b_3 \\ &p = a(p + b_2) = a(a b_3 + b_2) = a^2 b_3 + a b_2 \\ &p = a(p + b_1) = a(a^2 b_3 + a b_2 + b_1) = a^3 b_3 + a^2 b_2 + a b_1 = B(a) \end{aligned}$$

	natural binary		offset binary		2's C
$b_1 b_2 b_3 b_4$	m	$x_Q = Qm$	m'	$x_Q = Qm'$	$b_1 b_2 b_3 b_4$
—	16	8.0	8	4.0	—
1 1 1 1	15	7.5	7	3.5	0 1 1 1
1 1 1 0	14	7.0	6	3.0	0 1 1 0
1 1 0 1	13	6.5	5	2.5	0 1 0 1
1 1 0 0	12	6.0	4	2.0	0 1 0 0
1 0 1 1	11	5.5	3	1.5	0 0 1 1
1 0 1 0	10	5.0	2	1.0	0 0 1 0
1 0 0 1	9	4.5	1	0.5	0 0 0 1
1 0 0 0	8	4.0	0	0.0	0 0 0 0
0 1 1 1	7	3.5	−1	−0.5	1 1 1 1
0 1 1 0	6	3.0	−2	−1.0	1 1 1 0
0 1 0 1	5	2.5	−3	−1.5	1 1 0 1
0 1 0 0	4	2.0	−4	−2.0	1 1 0 0
0 0 1 1	3	1.5	−5	−2.5	1 0 1 1
0 0 1 0	2	1.0	−6	−3.0	1 0 1 0
0 0 0 1	1	0.5	−7	−3.5	1 0 0 1
0 0 0 0	0	0.0	−8	−4.0	1 0 0 0

Table P2.1 Converter codes for $B = 4$ bits, $R = 8$ volts.

Problem 2.14

The quantization width is $Q = R/2^B = 8/2^4 = 0.5$ volts. Table P2.1 shows the quantization levels and their binary codes.

To convert $x = 1.2$ by rounding, we shift it by half the quantization spacing $y = x + Q/2 = 1.2 + 0.25 = 1.45$. The following tables show the successive approximation tests for the natural and offset binary cases. The 2's complement case is obtained by complementing the MSB of the offset case.

test	$b_1 b_2 b_3 b_4$	x_Q	$C = u(y - x_Q)$
b_1	1 0 0 0	4.0	0
b_2	0 1 0 0	2.0	0
b_3	0 0 1 0	1.0	1
b_4	0 0 1 1	1.5	0
	0 0 1 0	1.0	

test	$b_1 b_2 b_3 b_4$	x_Q	$C = u(y - x_Q)$
b_1	1 0 0 0	0.0	1
b_2	1 1 0 0	2.0	0
b_3	1 0 1 0	1.0	1
b_4	1 0 1 1	1.5	0
	1 0 1 0	1.0	

The natural binary and 2'C cases agree because x lies in the positive range of both quantizers. For $x = 5.2$, we have $y = x + Q/2 = 5.45$. The tests are shown below:

test	$b_1 b_2 b_3 b_4$	x_Q	$C = u(y - x_Q)$
b_1	1 0 0 0	4.0	1
b_2	1 1 0 0	6.0	0
b_3	1 0 1 0	5.0	1
b_4	1 0 1 1	5.5	0
	1 0 1 0	5.0	

test	$b_1 b_2 b_3 b_4$	x_Q	$C = u(y - x_Q)$
b_1	1 0 0 0	0.0	1
b_2	1 1 0 0	2.0	1
b_3	1 1 1 0	3.0	1
b_4	1 1 1 1	3.5	1
	1 1 1 1	3.5	

For the 2'C case, the quantizer saturates to its maximum positive value. Finally, for $x = -3.2$, we have $y = -2.95$, and the following testing tables:

test	$b_1b_2b_3b_4$	x_Q	$C = u(y - x_Q)$
b_1	1 0 0 0	4.0	0
b_2	0 1 0 0	2.0	0
b_3	0 0 1 0	1.0	0
b_4	0 0 0 1	0.5	0
	0 0 0 0	0.0	

test	$b_1b_2b_3b_4$	x_Q	$C = u(y - x_Q)$
b_1	1 0 0 0	0.0	0
b_2	0 1 0 0	−2.0	0
b_3	0 0 1 0	−3.0	1
b_4	0 0 1 1	−2.5	0
	0 0 1 0	−3.0	

In this case, the natural binary scale saturates to its minimum positive value.

Problem 2.15

Similar to the testing tables of Examples 2.4.1 and 2.4.2.

Problem 2.16

These versions are straight forward. To implement truncation instead of rounding, replace the statement within adc:

```
y = x + Q/2;
```

by

```
y = x;
```

Problem 2.17

This problem is basically the same as Example 2.4.5. The same program segment of that example applies here for generating the codes and quantized values. Only the full-scale range is different, $R = 32$ volts. In part (c), the amplitude of the sinusoid $A = 20$ exceeds the quantizer range $R/2 = 16$, and therefore both the positive and negative peaks of the sinusoid will saturate to the positive and negative maxima of the quantizer.

Problem 2.18

The following C program carries out all the required operations. It uses the routine corr of Appendix A.1 to compute all the sample autocorrelations and cross-correlations.

```
/* exmpl246.c - quantization noise model
 *
 * default values:
 *     N=10 bits
 *     R=1024
 *     L=1000, iseed=10, M=50
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

void adc(), corr();
double dac();
```



```
double ran();

void main()
{
       int n, i, L, N, M, *b;
       long iseed;
       double *x, xq, *e, R, Q, *Ree, *Rex, *Rxx;
       double normee, normex;
       double v, pi = 4 * atan(1.0);

       FILE *fpe, *fpx;
       FILE *fpRee, *fpRex, *fpRxx;

       printf("enter N, R = ");
       scanf("%d %lf", &N, &R);
       printf("enter L, iseed, M = ");
       scanf("%d %ld %d", &L, &iseed, &M);

       fpe   = fopen("e",   "w");
       fpx   = fopen("x",   "w");

       fpRee = fopen("Ree",   "w");
       fpRex = fopen("Rex",   "w");
       fpRxx = fopen("Rxx",   "w");

       b = (int *) calloc(N, sizeof(int));

       x = (double *) calloc(L, sizeof(double));
       e = (double *) calloc(L, sizeof(double));

       Ree = (double *) calloc(M+1, sizeof(double));
       Rex = (double *) calloc(M+1, sizeof(double));
       Rxx = (double *) calloc(M+1, sizeof(double));

       Q = R / (1 << N);

       for (n=0; n<L; n++) {
              v    = 0.25 * R * (2 * ran(&iseed) - 1);
              x[n] = 0.25 * R * cos(2 * pi  * n / sqrt(131.)) + v;
              adc(x[n], b, N, R);
              xq = dac(b, N, R);
              e[n] = x[n] - xq;
              fprintf(fpx, "% lf\n", x[n]);
              fprintf(fpe, "% lf\n", e[n]);
              }

       corr(L, e, e, M, Ree);
       corr(L, e, x, M, Rex);
       corr(L, x, x, M, Rxx);

       normee = Ree[0];
       normex = sqrt(Ree[0] * Rxx[0]);

       for (i=0; i<=M; i++) {
              fprintf(fpRee, "%lf\n", Ree[i]/normee);
              fprintf(fpRex, "%lf\n", Rex[i]/normex);
              fprintf(fpRxx, "%lf\n", Rxx[i]/Rxx[0]);
              }
```

```
        printf("erms = Q/sqrt(12) = %lf\n", Q/sqrt(12.0));
        printf("Ree(0) = %lf\n", sqrt(normee));
}
```

Problem 2.19

The mean of v is zero because $E[u] = 0.5$:

$$E[v] = 0.5R(E[u] - 0.5) = 0$$

The variance of v is related to the variance of u as follows:

$$\sigma_v^2 = E[v^2] = (0.5R)^2 E[(u - 0.5)^2] = (0.5R)^2 \sigma_u^2 = (0.5R)^2 \frac{1}{12} = \frac{R^2}{48}$$

Problem 2.20

Inserting $x(n) = A\cos(2\pi f_0 n + \phi) + v(n)$ into the definition of the autocorrelation function, we find:

$$\begin{aligned} R_{xx}(k) &= E[x(n+k)x(n)] \\ &= A^2 E[\cos(2\pi f_0(n+k) + \phi)\cos(2\pi f_0 n + \phi)] \\ &+ AE[\cos(2\pi f_0(n+k) + \phi)v(n)] + AE[v(n+k)\cos(2\pi f_0 n + \phi)] \\ &+ E[v(n+k)v(n)] \end{aligned}$$

assuming ϕ and $v(n)$ are independent random variables, the cross terms will vanish. Indeed, by the independence assumption the expectation value of the product becomes the product of the expectation values:

$$E[v(n+k)\cos(2\pi f_0 n + \phi)] = E[v(n+k)] \cdot E[cos(2\pi f_0 n + \phi)] = 0$$

where the last zero follows either from the zero-mean property of $v(n)$ and/or from the zero mean property of $E[\cos(2\pi f_0 n + \phi)] = 0$. The latter follows from uniformity of ϕ whose probability density will be: $p(\phi) = 1/(2\pi)$. We have:

$$E[\cos(a + \phi)] = \int_0^{2\pi} \cos(a + \phi) p(\phi)\, d\phi = \frac{1}{2\pi} \int_0^{2\pi} \cos(a + \phi)\, d\phi = 0$$

Similarly, we have the property:

$$\begin{aligned} E[\cos(a + \phi)\cos(b + \phi)] &= \int_0^{2\pi} \cos(a + \phi)\cos(b + \phi) p(\phi)\, d\phi \\ &= \frac{1}{2\pi} \int_0^{2\pi} \cos(a + \phi)\cos(b + \phi)\, d\phi \\ &= \frac{1}{2}\cos(a - b) \end{aligned}$$

Thus, we find

$$E[\cos(2\pi f_0(n+k) + \phi)\cos(2\pi f_0 n + \phi)] = \frac{1}{2}\cos(2\pi f_0(n+k) - 2\pi f_0 n) = \frac{1}{2}\cos(2\pi f_0 k)$$

Finally, because $v(n)$ is assumed to be zero-mean white noise, we will have:

$$E[v(n+k)v(n)] = \sigma_v^2 \delta(k)$$

It follows that the autocorrelation of the noisy sinusoid will be:

$$R_{xx}(k) = \frac{1}{2}A^2 \cos(2\pi f_0 k) + \sigma_v^2 \delta(k)$$

At lag $k = 0$, we have:

$$R_{xx}(0) = \frac{1}{2}A^2 + \sigma_v^2$$

Therefore, the normalized autocorrelation function becomes:

$$\rho_{xx}(k) = \frac{R_{xx}(k)}{R_{xx}(0)} = \frac{A^2/2}{A^2/2 + \sigma_v^2}\cos(2\pi f_0 k) + \frac{\sigma_v^2}{A^2/2 + \sigma_v^2}\delta(k)$$

Defining the parameter:

$$a = \frac{A^2/2}{A^2/2 + \sigma_v^2} \qquad \Rightarrow \qquad 1 - a = \frac{\sigma_v^2}{A^2/2 + \sigma_v^2}$$

we finally have:

$$\rho_{xx}(k) = a\cos(2\pi f_0 k) + (1 - a)\delta(k)$$

Defining the signal to noise ratio as the relative strength of the sinusoid versus the noise in the autocorrelation function, that is,

$$SNR = \frac{A^2/2}{\sigma_v^2}$$

we may express the parameter a as:

$$a = \frac{SNR}{SNR + 1} \qquad \Rightarrow \qquad 1 - a = \frac{1}{SNR + 1}$$

Problem 2.21

The quantization width of the first ADC will be $Q_1 = R_1/2^{B_1}$. The residual signal formed at the output of the first DAC, $x_2 = x - x_1$, is the quantization error, and therefore, it varies of the limits

$$-\frac{Q_1}{2} \le x_2 \le \frac{Q_1}{2}$$

These limits must serve as the full scale range of the second ADC. Thus, we must choose:

$$R_2 = Q_1 = \frac{R_1}{2^{B_1}}, \qquad Q_2 = \frac{R_2}{2^{B_2}} = \frac{R_1}{2^{B_1+B_2}}$$

The following program segment illustrates the sequence of operations:

```
R2 = R1 / (1 << B1);

b1 = (int *) calloc(B1, sizeof(int));
b2 = (int *) calloc(B2, sizeof(int));

for (n=0; n<L; n++) {
       x = A * cos(2 * pi * f0 * n);
       adc(x, b1, B1, R1);
       x1 = dac(b1, B1, R1);
       x2 = x - x1;
       adc(x2, b2, B2, R2);
       }
```

When there is a third B_3-bit stage, the total number of bits will be

$$B = B_1 + B_2 + B_3$$

The B_2-bit output of the second ADC must be passed into a second DAC to get the corresponding analog value and subtracted from the input $x - x_1$ to get the third error output, which is then quantized to B_3 bits with a full scale range and quantization width:

$$R_3 = Q_2 = \frac{R_2}{2^{B_2}}, \qquad Q_3 = \frac{R_3}{2^{B_3}} = \frac{R_1}{2^{B_1+B_2+B_3}}$$

The following loop shows these operations:

```
R2 = R1 / (1 << B1);
R3 = R2 / (1 << B2);

b1 = (int *) calloc(B1, sizeof(int));
b2 = (int *) calloc(B2, sizeof(int));
b3 = (int *) calloc(B3, sizeof(int));

for (n=0; n<L; n++) {
       x = A * cos(2 * pi * f0 * n);
       adc(x, b1, B1, R1);
       x1 = dac(b1, B1, R1);
       adc(x-x1, b2, B2, R2);
       x2 = dac(b2, B2, R2);
       x3 = x - x1 - x2;
       adc(x3, b3, B3, R3);
       }
```

Problem 2.22

The quantized triangularly-dithered signals were generated by the for-loop given in Example 2.5.1.

Chapter 3 Problems

Problem 3.1

(a) If the input is doubled the output is not doubled. Thus, the system is not linear. Indeed, the output due to $x_1(n) = 2x(n)$ will be $y_1(n) = 3x_1(n)+5 = 6x(n)+5 \neq 2(3x(n)+5)$. The system is, however, time-invariant since it has constant-in-time coefficients.

(b) The quadratic term $x^2(n-1)$ breaks linearity because it quadruples whenever the input doubles. The term $x(2n)$ will break time-invariance. Indeed, let $x_D(n) = x(n-D)$ be a delayed input. The corresponding output $y_D(n)$ can be computed from the given I/O equation:

$$y_D(n) = x_D^2(n-1)+x_D(2n) = x^2(n-1-D)+x(2n-D)$$

but, this is not the same as the delayed (shifted) $y(n)$ obtained by replacing n by $n-D$:

$$y(n-D) = x^2(n-D-1)+x(2(n-D)) = x^2(n-D-1)+x(2n-2D)$$

Thus, the system is neither linear nor time-invariant.

(c) System is time-invariant, but not linear since the output does not double whenever the input doubles.

(d) The system is linear, but the term $nx(n-3)$ breaks time-invariance.

(e) The time-dependent term n breaks both time invariance and linearity.

Problem 3.2

Picking out the coefficients of the x-terms, we find:

$$(a) \qquad \mathbf{h} = [3, -2, 0, 4]$$

$$(b) \qquad \mathbf{h} = [4, 1, 0, -3]$$

$$(c) \qquad \mathbf{h} = [1, 0, 0, -1]$$

Problem 3.3

(a,b) These problems are of the general form:

$$y(n) = ay(n-1)+x(n)$$

with $a = -0.9$ for (a), and $a = 0.9$ for (b). Setting $x(n) = \delta(n)$ we find the difference equation for the impulse response:

$$h(n) = ah(n-1)+\delta(n)$$

Iterating forward in time with zero initial condition $h(-1) = 0$, we obtain the causal solution:

$$\begin{aligned}
&h(0) = ah(-1)+\delta(0) = a\cdot 0 + 1 = 1\\
&h(1) = ah(0)+\delta(1) = a\cdot 1 + 0 = a\\
&h(2) = ah(1)+\delta(2) = a\cdot a + 0 = a^2\\
&h(3) = ah(2)+\delta(3) = a\cdot a^2 + 0 = a^3\\
&\cdots\\
&h(n) = ah(n-1)+\delta(n) = a\cdot a^{n-1} = a^n, \quad \text{for } n \geq 1
\end{aligned}$$

Thus, $h(n)$ will be the geometric series:

$$\mathbf{h} = [1, a, a^2, a^3, \dots]$$

(c,d) These problems are similar to

$$y(n) = ay(n-2) + x(n)$$

with $a = 0.64 = (0.8)^2$ for (c), and $a = -0.81 = -(0.9)^2$ for (d). The impulse response will satisfy the difference equation:

$$h(n) = ah(n-2) + \delta(n)$$

A few iterations (with zero initial conditions) give:

$$\begin{aligned} h(0) &= 1 \\ h(1) &= 0 \\ h(2) &= ah(0) = a \\ h(3) &= ah(1) = 0 \\ h(4) &= ah(2) = a^2 \\ h(5) &= ah(3) = 0, \qquad \text{etc.} \end{aligned}$$

Thus, all the odd-indexed $h(n)$ are zero:

$$\mathbf{h} = [1, 0, a, 0, a^2, 0, a^3, 0, a^4, 0, \dots]$$

(e) The impulse response satisfies:

$$h(n) = 0.5h(n-1) + 4\delta(n) + \delta(n-1)$$

A few iterations give:

$$\begin{aligned} h(0) &= 4 \\ h(1) &= 0.5h(0) + 1 = 3 \\ h(2) &= 0.5h(1) = 3(0.5) \\ h(3) &= 0.5h(2) = 3(0.5)^2, \quad \text{etc.} \end{aligned}$$

Thus,

$$h(n) = \begin{cases} 4 & \text{if } n = 0 \\ 3(0.5)^{n-1} & \text{if } n \geq 1 \end{cases}$$

Problem 3.4

We have seen that if $h(n) = a^n u(n)$, then the I/O difference equation is

$$y(n) = ay(n-1) + x(n)$$

In cases (a) and (b), we apply this result with $a = 0.9$ and $a = -0.9$. In cases (c) and (d), we apply this result *twice* and verify that $h(n) = a^n u(n) + (-a)^n u(n)$ leads to the difference equation

$$y(n) = a^2 y(n-2) + 2x(n)$$

Thus, for (c) we have $a = 0.9$, $a^2 = 0.81$, and

$$y(n) = 0.81y(n-2) + 2x(n)$$

And, for (d) we have $a = 0.9j$, $a^2 = -0.81$, and

$$y(n) = -0.81y(n-2) + 2x(n)$$

A systematic and simpler way of solving this type of problem will be presented after we cover z-transforms.

Problem 3.5

The impulse response sequence is explicitly:

$$\mathbf{h} = [4, 3, 3(0.5), 3(0.5)^2, 3(0.5)^3, \dots]$$

Replacing these values into the convolutional equation, we get:

$$\begin{aligned} y_n &= h_0 x_n + h_1 x_{n-1} + h_2 x_{n-2} + h_3 x_{n-3} + h_4 x_{n-4} + \cdots \\ &= 4x_n + 3\left[x_{n-1} + 0.5x_{n-2} + 0.5^2 x_{n-3} + 0.5^3 x_{n-4} + \cdots\right] \end{aligned}$$

It follows that:

$$0.5y_{n-1} = 2x_{n-1} + 3\left[0.5x_{n-2} + 0.5^2 x_{n-3} + 0.5^3 x_{n-4} \cdots\right]$$

Subtracting, we have:

$$y_n - 0.5y_{n-1} = 4x_n + 3x_{n-1} - 2x_{n-1} \qquad \text{or,}$$

$$y_n = 0.5y_{n-1} = 4x_n + x_{n-1}$$

Problem 3.6

We may write the given impulse response in the form:

$$h(n) = 5\delta(n) + 6(0.8)^{n-1}u(n-1) = [5, 6, 6(0.8), 6(0.8)^2, \dots]$$

Proceeding as in the previous problem we have:

$$\begin{aligned} y_n &= 5x_n + 6\left[x_{n-1} + 0.8x_{n-2} + 0.8^2 x_{n-3} + \cdots\right] \\ 0.8y_{n-1} &= 4x_{n-1} + 6\left[0.8x_{n-2} + 0.8^2 x_{n-3} + \cdots\right] \end{aligned}$$

which gives

$$y_n - 0.8y_{n-1} = 5x_n + 2x_{n-1}$$

Problem 3.7

It is evident from the definition that for $n \geq 2$, $h(n)$ satisfies the recursions $h(n) = ah(n-1)$. Instead of manipulating the I/O convolutional equation into a difference equation as we did above, let us determine directly the difference equation satisfied by $h(n)$ and from that determine the difference equation for $y(n)$.

The given expression for $h(n)$ can be rewritten in the more compact form, which is valid for all n:

$$h(n) = c_0\delta(n) + c_1 a^{n-1}u(n-1)$$

where the shifted unit-step $u(n-1)$ vanishes for $n = 0$ and equals one for all $n \geq 1$. Because $h(n)$ satisfies $h(n) = ah(n-1)$ for $n \geq 2$, we are led to consider the delayed function $h(n-1)$ and multiply it by a:

$$h(n-1) = c_0\delta(n-1) + c_1 a^{n-2}u(n-2)$$

and

$$ah(n-1) = ac_0\delta(n-1) + c_1 a^{n-1}u(n-2)$$

Subtracting, we have

$$h(n) - ah(n-1) = c_0\delta(n) - ac_0\delta(n-1) + c_1 a^{n-1}\big[u(n-1) - u(n-2)\big]$$

But the difference $u(n-1) - u(n-2) = \delta(n-1)$ as follows from the standard result $u(n) - u(n-1) = \delta(n)$, which was essentially derived in Example 3.4.4. Therefore, we have

$$h(n) - ah(n-1) = c_0\delta(n) - ac_0\delta(n-1) + c_1 a^{n-1}\delta(n-1)$$

But note that $a^{n-1}\delta(n-1) = \delta(n-1)$ because it is nonzero only at $n = 1$ for which we have $a^{1-1}\delta(0) = a^0\delta(0) = \delta(0)$. Then,

$$h(n) - ah(n-1) = c_0\delta(n) - ac_0\delta(n-1) + c_1\delta(n-1) = c_0\delta(n) + (c_1 - ac_0)\delta(n-1)$$

Or, setting $b_0 = c_0$ and $b_1 = c_1 - ac_0$ and solving for $h(n)$ we find

$$h(n) = ah(n-1) + b_0\delta(n) + b_1\delta(n-1)$$

Example 3.4.7 had $b_0 = c_0 = 2$ and $b_1 = c_1 - ac_0 = 4 - 2 \cdot 0.5 = 3$. Next, we map this difference equation for $h(n)$ into a difference equation for $y(n)$. To do this we start with the convolutional equation and replace $h(m)$ by its difference equation expression, that is,

$$\begin{aligned} y(n) &= \sum_m h(m)x(n-m) \\ &= \sum_m \big[ah(m-1) + b_0\delta(m) + b_1\delta(m-1)\big]x(n-m) \\ &= a\sum_m h(m-1)x(n-m) + \sum_m \big[b_0\delta(m) + b_1\delta(m-1)\big]x(n-m) \end{aligned}$$

In the second sum, the presence of $\delta(m)$ extracts only the $m = 0$ term, that is, $b_0x(n)$, whereas the presence of $\delta(m-1)$ extracts only the $m = 1$ term, that is, $b_1x(n-1)$. Moreover, with a change of variables of summation from m to $k = m-1$ or $m = k+1$, the a-term is recognized to be $ay(n-1)$; indeed,

$$a\sum_m h(m-1)x(n-m) = a\sum_k h(k)x(n-k-1) = ay(n-1)$$

the last equation following from the convolution equation by replacing n by $n-1$:

$$y(n) = \sum_k h(k)x(n-k) \qquad \Rightarrow \qquad y(n-1) = \sum_k h(k)x(n-1-k)$$

We finally find the I/O difference equation for $y(n)$:

$$y(n) = ay(n-1) + b_0x(n) + b_1x(n-1)$$

which is exactly the same as that satisfied by $h(n)$.

Problem 3.8

First, note that if $f_n = a^nu(n)$ then for $n \geq 1$

$$(1 - aD)f_n = f_n - af_{n-1} = a^n - aa^{n-1} = 0$$

where D is the delay operator defined by $(Df)_n = f_{n-1}$. It follows that the first term $p_1^nu(n)$ of h_n will be annihilated by $(1 - p_1D)$, the second term of h_n will be annihilated by $(1 - p_2D)$, etc. Therefore, all the terms in h_n will be annihilated by the product

$$(1 - p_1D)(1 - p_2D)\cdots(1 - p_MD) = 1 + a_1D + a_2D^2 + \cdots + a_MD^M$$

Problem 3.9

The $C_ip_i^n$ terms are annihilated by the same product as in the previous problem. However, the $\delta(n)$ term will contain the delayed terms:

$$a_M\delta(n-M) + a_{M-1}\delta(n-M+1) + \cdots$$

But, if $n \geq M+1$, then these terms vanish too. Thus, the difference equation satisfied by h_n will be:

$$h_n + a_1h_{n-1} + \cdots + a_Mh_{n-M} = 0, \qquad n \geq M+1$$

Problem 3.10

Define the polynomial with roots p_1, p_2:

$$1 + a_1z^{-1} + a_2z^{-2} = (1 - p_1z^{-1})(1 - p_2z^{-1})$$

so that

$$a_1 = -(p_1 + p_2), \quad a_2 = p_1p_2$$

Then, the p_i^n terms in h_n will be annihilated by the delay polynomial:

$$h_n + a_1h_{n-1} + a_2h_{n-2} = (1 + a_1D + a_2D^2)h_n = (1 - p_1D)(1 - p_2D)h_h$$

for $n \geq 2$. The coefficients b_0, b_1 may be found by explicitly calculating the right-hand side of this difference equation using the expression for h_n, for $n = 0, 1$:

$$b_0 = h_0 + a_1 h_{-1} + a_2 h_{-2} = h_0 = C_1 + C_2$$

$$b_1 = h_1 + a_1 h_0 + a_2 h_{-1} = h_1 + a_1 h_0 = C_1 p_1 + C_2 p_2 + a_1 (C_1 + C_2)$$

Using the expression for a_1, we find:

$$b_0 = C_1 + C_2$$

$$b_1 = C_1 p_1 + C_2 p_2 - (p_1 + p_2)(C_1 + C_2) = -(C_1 p_2 + C_2 p_1)$$

To summarize, h_n will satisfy the difference equation:

$$h_n + a_1 h_{n-1} + a_2 h_{n-2} = b_0 \delta(n) + b_1 \delta(n-1) \tag{P3.1}$$

Inserting this into the convolutional equation, we obtain the I/O difference equation:

$$y_n + a_1 y_{n-1} + a_2 y_{n-2} = b_0 x_n + b_1 x_{n-1}$$

These results can be obtained quickly using z-transforms:

$$H(z) = \frac{C_1}{1 - p_1 z^{-1}} + \frac{C_2}{1 - p_2 z^{-1}} = \frac{(C_1 + C_2) - (C_1 p_2 + C_2 p_1) z^{-1}}{(1 - p_1 z^{-1})(1 - p_2 z^{-1})}$$

which may be rewritten as

$$H(z) = \frac{b_0 + b_1 z^{-1}}{1 + a_1 z^{-1} + a_2 z^{-2}}$$

and leads to the desired difference equations for h_n and y_n.

Problem 3.11

Now the difference equation Eq. (P3.1) will be satisfied for $n \geq 3$. To determine the coefficients b_0, b_1, b_2, we evaluate the right hand side for $n = 0, 1, 2$:

$$b_0 = h_0$$

$$b_1 = h_1 + a_1 h_0$$

$$b_2 = h_2 + a_1 h_1 + a_2 h_0$$

which give:

$$b_0 = C_0 + C_1 + C_2$$

$$b_1 = C_1 p_1 + C_2 p_2 + a_1 (C_0 + C_1 + C_2) = -(p_1 + p_2) C_0 - (C_1 p_2 + C_2 p_1)$$

$$b_2 = C_1 p_1^2 + C_2 p_2^2 + a_1 (C_1 p_1 + C_2 p_2) + a_2 (C_0 + C_1 + C_2) = C_0 p_1 p_2$$

In b_2, the coefficient of C_1 is $p_1^2 + a_1 p_1 + a_2$, which is zero because p_1 is a root of the polynomial $z^2 + a_1 z + a_2$. Similarly, the coefficient of C_2 is zero.
Again, the results can be justified quickly using z-transforms:

$$H(z) = C_0 + \frac{C_1}{1 - p_1 z^{-1}} + \frac{C_2}{1 - p_2 z^{-1}}$$

$$= \frac{(C_0 + C_1 + C_2) - (C_1 p_2 + C_2 p_1 + C_0 (p_1 + p_2)) z^{-1} + C_0 p_1 p_2 z^{-2}}{(1 - p_1 z^{-1})(1 - p_2 z^{-1})}$$

Problem 3.12

Define the polynomial with roots p_1, p_2, p_3:

$$1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} = (1 - p_1 z^{-1})(1 - p_2 z^{-1})(1 - p_3 z^{-1})$$

so that

$$a_1 = -(p_1 + p_2 + p_3), \quad a_2 = p_1 p_2 + p_2 p_3 + p_3 p_1, \quad a_3 = -p_1 p_2 p_3$$

Then, the p_i^n terms in h_n will be annihilated by the delay polynomial:

$$(1 + a_1 D + a_2 D^2 + a_3 D^3) h_n = (1 - p_1 D)(1 - p_2 D)(1 - p_3 D) h_h = 0$$

for $n \geq 3$, or,

$$h_n + a_1 h_{n-1} + a_2 h_{n-2} + a_3 h_{n-3} = 0, \qquad n \geq 3$$

The coefficients b_0, b_1, b_2 may be found by explicitly calculating the right-hand side of this difference equation using the expression for h_n, for $n = 0, 1, 2$:

$$b_0 = h_0$$

$$b_1 = h_1 + a_1 h_0$$

$$b_2 = h_2 + a_1 h_1 + a_2 h_0$$

or,

$$b_0 = C_1 + C_2 + C_3$$

$$b_1 = C_1 p_1 + C_2 p_2 + C_3 p_3 + a_1 (C_1 + C_2 + C_3)$$

$$b_2 = C_1 p_1^2 + C_2 p_2^2 + C_3 p_3^2 + a_1 (C_1 p_1 + C_2 p_2 + C_3 p_3) + a_2 (C_1 + C_2 + C_3)$$

which simplify into:

$$b_0 = C_1 + C_2 + C_3$$

$$b_1 = -[C_1 (p_2 + p_3) + C_2 (p_3 + p_1) + C_3 (p_1 + p_2)]$$

$$b_2 = C_1 p_2 p_3 + C_2 p_3 p_1 + C_3 p_1 p_2$$

Problem 3.13

For (a), we have with $n \geq 2$:

$$(1 - 0.5D)(1 - 0.8D) = 1 - 1.3D + 0.4D^2 \quad \Rightarrow$$

$$h_n - 1.3 h_{n-1} + 0.4 h_{n-2} = 0$$

For (b), we have with $n \geq 2$:

$$(1 - 0.5jD)(1 + 0.5jD) = 1 + 0.25D^2 \quad \Rightarrow$$

$$h_n + 0.25 h_{n-2} = 0$$

For (c), we have with $n \geq 3$:

$$(1 - 0.4D)(1 - 0.5D)(1 + 0.5D) = 1 - 0.4D - 0.25D^2 + 0.1D^3 \quad \Rightarrow$$

$$h_n - 0.4 h_{n-1} - 0.25 h_{n-2} + 0.1 h_{n-3} = 0$$

Problem 3.14

From the convolutional I/O equation we get:

$$|y(n)| = |\sum_m h(m)x(n-m)| \le \sum_m |h(m)||x(n-m)| \le \sum_m |h(m)|B = AB$$

Problem 3.15

Starting with

$$y(n) = \sum_m h(m)x(n-m)$$

we obtain at $n = 0$:

$$y(0) = \sum_m h(m)x(-m)$$

But, $x(m) = \text{sign}\big(h(-m)\big)$. Therefore, $x(-m) = \text{sign}\big(h(m)\big)$. It follows:

$$y(0) = \sum_m h(m)x(-m) = \sum_m h(m)\,\text{sign}\big(h(m)\big)$$

Recognizing that in general $x\,\text{sign}(x) = |x|$, we find

$$y(0) = \sum_m |h(m)|$$

Because $x(n)$ was bounded, $y(n)$ and, hence $y(0)$, must be bounded. If at some m, $h(m) = 0$, then this term would not contribute to the convolutional sum.

Problem 3.16

Working with the convolutional I/O equation, we have:

$$\begin{aligned} y_D(n) &= \sum_m h_D(m)x(n-m) = \sum_m h(m-D)x(n-m) \\ &= \sum_k h(k)x(n-D-k) = y(n-D) \end{aligned}$$

where we changed summation variables from m to $k = m - D$, and evaluated $y(n)$ at $n - D$, that is,

$$y(n) = \sum_k h(k)x(n-k) \quad \Rightarrow \quad y(n-D) = \sum_k h(k)x(n-D-k)$$

Problem 3.17

We have:

$$\sum_m h_D(m)x_A(n-m) = \sum_m h(m-D)x(n-m+D) = \sum_k h(k)x(n-k)$$

where we changed variables of summation from m to $k = m - D$.

Problem 3.18

Because $\tilde{h}(n)$ agrees with $h(n)$ for $n \ge -D$, we have:

$$y(n) = \sum_{m=-\infty}^{\infty} h(m)x(n-m)$$

$$\tilde{y}(n) = \sum_{m=-D}^{\infty} \tilde{h}(m)x(n-m) = \sum_{m=-D}^{\infty} h(m)x(n-m)$$

Subtracting, the terms $m \ge -D$ cancel and we get:

$$y(n) - \tilde{y}(n) = \sum_{m=-\infty}^{-D-1} h(m)x(n-m)$$

Assuming $x(n)$ is bounded by A, we have:

$$|y(n) - \tilde{y}(n)| = |\sum_{m=-\infty}^{-D-1} h(m)x(n-m)| \le \sum_{m=-\infty}^{-D-1} |h(m)||x(n-m)| \le \sum_{m=-\infty}^{-D-1} |h(m)|A$$

This summation is finite because it is a subsum of the stability condition:

$$\sum_{m=-\infty}^{-D-1} |h(m)| \le \sum_{m=-\infty}^{\infty} |h(m)| < \infty$$

In the limit of large D, the number of terms shrinks to zero giving:

$$\lim_{D\to\infty} \sum_{m=-\infty}^{-D-1} |h(m)| \to 0$$

Chapter 4 Problems

Problem 4.1

The convolution table is:

h**x**	1	2	1	1	2	1	1	1
1	1	2	1	1	2	1	1	1
1	1	2	1	1	2	1	1	1
2	2	4	2	2	4	2	2	2
1	1	2	1	1	2	1	1	1

Folding the table, we get

$$\mathbf{y} = [1,\ 3,\ 5,\ 7,\ 7,\ 6,\ 7,\ 6,\ 4,\ 3,\ 1]$$

The first and last three samples are the input and output transients, and the middle 5 samples are the steady-state outputs. The LTI table is:

n	0	1	2	3	4	5	6	7	8	9	10	
x**h**	1	1	2	1								partial output
1	1	1	2	1								$x(0)h(n-0)$
2		2	2	4	2							$x(1)h(n-1)$
1			1	1	2	1						$x(2)h(n-2)$
1				1	1	2	1					$x(3)h(n-3)$
2					2	2	4	2				$x(4)h(n-4)$
1						1	1	2	1			$x(5)h(n-5)$
1							1	1	2	1		$x(6)h(n-6)$
1								1	1	2	1	$x(7)h(n-7)$
$y(n)$	1	3	5	7	7	6	7	6	4	3	1	$\sum_m x(m)h(n-m)$

For the overlap-add method, the input is divided into the following three contiguous blocks:

$$\mathbf{x} = [\underbrace{1,2,1}_{\mathbf{x}_0},\underbrace{1,2,1}_{\mathbf{x}_1},\underbrace{1,1,0}_{\mathbf{x}_2}]$$

where we padded an extra zero at the end to get a length-3 block. Convolving each block separately with **h** gives:

$$\mathbf{y}_0 = \mathbf{h} * \mathbf{x}_0 = [1,3,5,6,4,1]$$

$$\mathbf{y}_1 = \mathbf{h} * \mathbf{x}_1 = [1,3,5,6,4,1]$$

$$\mathbf{y}_2 = \mathbf{h} * \mathbf{x}_2 = [1,2,3,3,1,0]$$

These convolutions can be done by separately folding the three convolution subtables:

	block 0			block 1			block 2		
h**x**	1	2	1	1	2	1	1	1	0
1	1	2	1	1	2	1	1	1	0
1	1	2	1	1	2	1	1	1	0
2	2	4	2	2	4	2	2	2	0
1	1	2	1	1	2	1	1	1	0

The three subblocks begin at the absolute times $n = 0, 3, 6$, respectively. It follows from time-invariance that the corresponding output blocks will also begin at the same absolute times. Thus, aligning the output blocks according to their absolute timings and adding them up gives the final result:

n	0	1	2	3	4	5	6	7	8	9	10
$\mathbf{y}_0$	1	3	5	6	4	1					
$\mathbf{y}_1$				1	3	5	6	4	1		
$\mathbf{y}_2$							1	2	3	3	1
$\mathbf{y}$	1	3	5	7	7	6	7	6	4	3	1

In practice this method is implemented efficiently by computing the individual block convolutions using the FFT instead of time-domain convolution. For an FIR filter of order M and an FFT of length N (which is a power of two), the length of each x-block is chosen to be $N_1 = N - M$. The computational gain of this "fast" convolution method versus the conventional time-domain "slow" method is approximately

$$\frac{\text{fast}}{\text{slow}} = \frac{\log_2 N}{M}$$

If the input is divided into length-5 subblocks, the last subblock will have length 3:

$$\mathbf{x} = [\underbrace{1,2,1,1,2}_{\mathbf{x}_0},\underbrace{1,1,1}_{\mathbf{x}_1}]$$

The convolutions of the subblocks with the filter are:

$$\mathbf{y}_0 = \mathbf{h} * \mathbf{x}_0 = [1,3,5,7,7,5,5,2]$$

$$\mathbf{y}_1 = \mathbf{h} * \mathbf{x}_1 = [1,2,4,4,3,1]$$

Aligning them at multiples of $n = 5$, we have:

n	0	1	2	3	4	5	6	7	8	9	10
$\mathbf{y}_0$	1	3	5	7	7	5	5	2			
$\mathbf{y}_1$						1	2	4	4	3	1
$\mathbf{y}$	1	3	5	7	7	6	7	6	4	3	1

Problem 4.2

The convolution table is:

h**x**	2	2	0	1	-1	0	1	2
2	4	4	0	2	-2	0	2	4
-2	-4	-4	0	-2	2	0	-2	-4
-1	-2	-2	0	-1	1	0	-1	-2
1	2	2	0	1	-1	0	1	2

Folding the table, we get

$$\mathbf{y} = [4,\ 0,\ -6,\ 2,\ -2,\ 1,\ 4,\ 1,\ -5,\ -1,\ 2]$$

The first and last three samples are the input and output transients, and the middle 5 samples are the steady-state outputs. The LTI table is:

n	0	1	2	3	4	5	6	7	8	9	10	
$\mathbf{x}\backslash\mathbf{h}$	2	-2	-1	1								partial output
2	4	-4	-2	2								$x(0)h(n-0)$
2		4	-4	-2	2							$x(1)h(n-1)$
0			0	0	0	0						$x(2)h(n-2)$
1				2	-2	-1	1					$x(3)h(n-3)$
-1					-2	2	1	-1				$x(4)h(n-4)$
0						0	0	0	0			$x(5)h(n-5)$
1							2	-2	-1	1		$x(6)h(n-6)$
2								4	-4	-2	2	$x(7)h(n-7)$
$y(n)$	4	0	-6	2	-2	1	4	1	-5	-1	2	$\sum_m x(m)h(n-m)$

For the overlap-add method, the input is divided into the following three contiguous blocks:

$$\mathbf{x} = [\underbrace{2,2,0}_{\mathbf{x}_0},\underbrace{1,-1,0}_{\mathbf{x}_1},\underbrace{1,2,0}_{\mathbf{x}_2}]$$

where we padded an extra zero at the end to get a length-3 block. Convolving each block separately with $\mathbf{h}$ gives:

$$\mathbf{y}_0 = \mathbf{h} * \mathbf{x}_0 = [4,0,-6,0,2]$$

$$\mathbf{y}_1 = \mathbf{h} * \mathbf{x}_1 = [2,-4,1,2,-1,0]$$

$$\mathbf{y}_2 = \mathbf{h} * \mathbf{x}_2 = [2,-2,-5,-1,2]$$

These convolutions can be done by separately folding the three convolution subtables:

	block 0			block 1			block 2	
$\mathbf{h}\backslash\mathbf{x}$	2	2	0	1	-1	0	1	2
2	4	4	0	2	-2	0	2	4
-2	-4	-4	0	-2	2	0	-2	-4
-1	-2	-2	0	-1	1	0	-1	-2
1	2	2	0	1	-1	0	1	2

Aligning the output subblocks at multiples of $n = 3$, we get:

n	0	1	2	3	4	5	6	7	8	9	10
$\mathbf{y}_0$	4	0	-6	0	2	1					
$\mathbf{y}_1$				2	-4	1	2	-1	0		
$\mathbf{y}_2$							2	2	-5	-1	2
$\mathbf{y}$	4	0	-6	2	-2	1	4	1	-5	-1	2

If the input is divided into length-5 subblocks, the last subblock will have length 3:

$$\mathbf{x} = [\underbrace{2,2,0,1,-1}_{\mathbf{x}_0},\underbrace{0,1,2}_{\mathbf{x}_1}]$$

The convolutions of the subblocks with the filter are:

$$\mathbf{y}_0 = \mathbf{h} * \mathbf{x}_0 = [4,0,-6,2,-2,1,2,-1]$$

$$\mathbf{y}_1 = \mathbf{h} * \mathbf{x}_1 = [0,2,2,-5,-1,2]$$

Aligning them at multiples of $n = 5$, we have:

n	0	1	2	3	4	5	6	7	8	9	10
$\mathbf{y}_0$	4	0	-6	2	-2	1	2	-1			
$\mathbf{y}_1$						0	2	2	-5	-1	2
$\mathbf{y}$	4	0	-6	2	-2	1	4	1	-5	-1	2

Problem 4.3

Figure P4.1 shows the filter, input, and computed output signals.

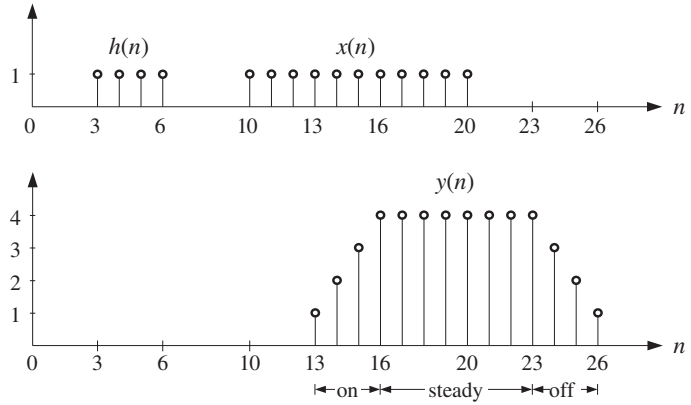


Fig. P4.1 $h(n)$, $x(n)$, and $y(n)$ signals of Problem 4.3.

For the direct form, the indices of h_m and x_{n-m} are restricted to lie within their respective ranges:

$$\begin{aligned} 3 \le m \le 6 \\ 10 \le n-m \le 20 \end{aligned} \tag{P4.1}$$

Solving the second with respect to n, we have:

$$10 + m \le n \le 20 + m$$

Using the first of Eq. (P4.1), we extend the range to:

$$10 + 3 \le 10 + m \le n \le 20 + m \le 20 + 6 \quad \Rightarrow \quad \boxed{13 \le n \le 26}$$

This is the range of index n of the output y_n. To find the limits of summation over m, we change the sign of the second of Eq. (P4.1) and solve for m:

$$-20 \le m - n \le -10 \quad \Rightarrow \quad n - 20 \le m \le n - 10$$

Thus, the inequalities in Eq. (P4.1) are equivalent to

$$\begin{aligned} 3 \le m \le 6 \\ n - 20 \le m \le n - 10 \end{aligned} \quad \Rightarrow \quad \boxed{\max(3, n-20) \le m \le \min(6, n-10)}$$

The final expression for the direct form will be:

$$y_n = \sum_{m=\max(3,n-20)}^{\min(6,n-10)} h_m x_{n-m}\,, \quad \text{for} \quad 13 \le n \le 26$$

The transition from the input-on transients to the steady state takes place at time $6 = n - 10$ or $n = 16$ at which the upper limit switches from $n - 10$ to 6. Similarly, the transition from steady state to the input-off transients occurs at $3 = n - 20$ or $n = 23$ when the lower limit switches from 3 to $n - 10$. In summary, the three ranges are:

$$\text{input-on:} \quad 13 \le n \le 15 \qquad y_n = \sum_{m=3}^{n-10} h_m x_{n-m}$$

$$\text{steady-state:} \quad 16 \le n \le 23 \qquad y_n = \sum_{m=3}^{6} h_m x_{n-m}$$

$$\text{input-off:} \quad 24 \le n \le 26 \qquad y_n = \sum_{m=n-20}^{6} h_m x_{n-m}$$

In particular, if the filter and input signals are unity over the ranges (P4.1), the output can be expressed compactly as:

$$y_n = \sum_{m=\max(3,n-20)}^{\min(6,n-10)} 1 \cdot 1 = \min(6, n-10) - \max(3, n-20) + 1 \tag{P4.2}$$

for $13 \le n \le 26$. Or, more explicitly:

$$\text{input-on:} \quad 13 \le n \le 15 \qquad y_n = n - 10 - 3 + 1 = n - 12$$

$$\text{steady-state:} \quad 16 \le n \le 23 \qquad y_n = 6 - 3 + 1 = 4$$

$$\text{input-off:} \quad 24 \le n \le 26 \qquad y_n = 6 - (n-20) + 1 = 27 - n$$

With numerical values:

$$\mathbf{y} = \{1, 2, 3, 4, 4, 4, 4, 4, 4, 4, 4, 3, 2, 1\}$$

The first and last $M = 3$ samples are the input on/off transients. The middle samples are the steady samples. Note that because the input is constant, the steady-state DC output can be predicted in advance from the DC gain:

$$y_{\text{DC}} = \sum_m h_m = 1 + 1 + 1 + 1 = 4$$

Note also that because the filter is delayed by 3 samples, the output will not begin to come out until 3 time instants after the input begins, that is at $n = 13$. Also, the input is cutoff at $n = 20$ but this is not felt by the filter until 3 instants later, that is, at $n = 23$. For the LTI form, Eq. (P4.1) is replaced by

$$10 \le m \le 20$$

$$3 \le n - m \le 6$$

This leads to the same range for the output index n, but to the summation limits:

$$y_n = \sum_{m=\max(10,n-6)}^{\min(20,n-3)} x_m h_{n-m}\,, \quad \text{for} \quad 13 \le n \le 26$$

For part (b), we have:

$$y_n = \sum_{m=\max(10,n-6)}^{\min(20,n-3)} 1 \cdot 1 = \min(10, n-6) - \max(20, n-3) + 1 \tag{P4.3}$$

Note that Eq. (P4.3) is equivalent to Eq. (P4.2).

Problem 4.4

Because both the input and filter are causal and have infinite duration, we use the formula:

$$y(n) = \sum_{m=0}^{n} h(m)x(n-m) = \sum_{m=0}^{n} a^m u(m)u(n-m) \qquad \text{or,}$$

$$y(n) = \sum_{m=0}^{n} a^m = \frac{1 - a^{n+1}}{1 - a}$$

The steady-state response is the large-n limit of this formula; that is,

$$y(n) \longrightarrow \frac{1}{1-a} \quad \text{for} \quad n \longrightarrow \infty$$

If $x(n) = (-1)^n u(n)$, we have

$$y(n) = \sum_{m=0}^{n} a^m (-1)^{n-m} = (-1)^n \sum_{m=0}^{n} (-a)^m = (-1)^n \frac{1 - (-a)^{n+1}}{1 + a}$$

Problem 4.5

Consider the IIR filter $h(n) = a^n u(n)$, where $0 < a < 1$. The square pulse $x(n) = u(n) - u(n-L)$ of duration L is applied as input. Using the time-domain convolution formula determine a closed-form expression for the output signal $y(n)$ for the two time ranges: $0 \le n \le L-1$ and $n \ge L$. Because the filter is infinite ($M = \infty$), Eq. (4.1.16) becomes:

$$y(n) = \sum_{m=\max(0,n-L+1)}^{n} h(m)x(n-m), \qquad 0 \le n < \infty$$

Inserting the given signals for $h(n)$ and $x(n)$, we find:

$$y(n) = \sum_{m=\max(0,n-L+1)}^{n} a^m \cdot 1$$

Using the geometric series identity

$$\sum_{m=M_1}^{M_2} a^m = \frac{a^{M_1} - a^{M_2+1}}{1-a}$$

we find:

$$y(n) = \frac{a^{\max(0,n-L+1)} - a^{n+1}}{1-a}$$

In particular, for $0 \le n \le L-1$, we have $\max(0, n-L+1) = 0$, and

$$y(n) = \frac{1-a^{n+1}}{1-a} \qquad \text{(input-on transients and steady state)}$$

whereas for $n \ge L$, we have $\max(0, n-L+1) = n-L+1$, and

$$y(n) = \frac{a^{n-L+1} - a^{n+1}}{1-a} = a^{n-L+1}\frac{1-a^L}{1-a} = a^{n-L+1}y(L-1), \qquad \text{(input-off)}$$

Thus, after the input is turned off, the output decays exponentially to zero.

Problem 4.6

From the previous problem, we have at $n = 0$,

$$y(0) - ay(-1) = \frac{1-a^{0+1}}{1-a} - a \cdot 0 = 1 = x(0)$$

Then, for $1 \le n \le L-1$, we have $x(n) = 1$ and

$$y(n) - ay(n-1) = \frac{1-a^{n+1}}{1-a} - a\frac{1-a^n}{1-a} = \frac{1-a^{n+1}-a+a^{n+1}}{1-a} = 1 = x(n)$$

Finally, for $n \ge L$, we have $x(n) = 0$ and

$$y(n) - ay(n-1) = a^{n-L+1}y_{L-1} - aa^{n-1-L+1}y_{L-1} = 0 = x(n)$$

Problem 4.7

For example, the LTI form is obtained by interchanging the roles played by x and h and their lengths L and $M+1$. The following routine is an implementation of Eq. (4.1.19):

```
/* convlti.c - convolution in LTI form */

#include <stdlib.h>                          defines max( ) and min( )

void convlti(M, h, L, x, y)
double *h, *x, *y;                           h=filter, x=input block, y=output block
int M, L;                                    M=filter order, L=input length
{
       int n, m;

       for (n = 0; n < L+M; n++)
              for (y[n] = 0, m = max(0, n-M); m <= min(n, L-1); m++)
                     y[n] += x[m] * h[n-m];
}
```


Problem 4.8

The following C program computes the output signals of the two filters (a) and (b) of Example 4.1.8:

```
/* convex.c - convolution example 4.1.8 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define Ma 14
#define Mb  5

void conv();

void main()
{
       int K = 50, L = 200, n;
       double *x, *ya, *yb;
       double ha[Ma+1], hb[Mb+1] = {0.2, -1, 2, -2, 1, -0.2};

       x  = (double *) calloc(L, sizeof(double));
       ya = (double *) calloc(L+Ma, sizeof(double));      Ex. 4.1.8 (a)
       yb = (double *) calloc(L+Mb, sizeof(double));      Ex. 4.1.8.(b)

       for (n=0; n<=Ma; n++)                              define filter (a)
              ha[n] = 0.25 * pow(0.75, n);

       for (n=0; n<L; n++)                                define input
              if (n%K < K/2)
                     x[n] = 1;
              else
                     x[n] = 0;

       conv(Ma, ha, L, x, ya);                            compute outputs
       conv(Mb, hb, L, x, yb);

       for (n=0; n<L+Ma; n++)
              printf("%lf\n", ya[n]);

              printf("\n\n");

       for (n=0; n<L+Mb; n++)
              printf("%lf\n", yb[n]);
}
```

Problem 4.9

A complete such program is included below.

```
/* blkfilt.c - FIR filtering by block convolution */

#include <stdio.h>
#include <stdlib.h>

#define MAX 64                                        memory allocation size
```

```
void blockcon();

void main(int argc, char **argv)
{
    FILE *fph;                                          filter file
    double *h, *x, *y, *ytemp;
    int M, N, L, i;                                     M = filter order, L = blocksize
    int max = MAX, dmax = MAX;                          initial allocation & increment

    if (argc != 3) {
        fprintf(stderr, "usage: blkfilt hfile L <xfile >yfile\n");
        exit(0);
        }

    if ((fph = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "can't open filter file: %s\n", argv[1]);
        exit(0);
        }

    L = atoi(argv[2]);                                                  block length

    h = (double *) calloc(max + 1, sizeof(double));

    for (M=0;; M++) {                                   read h
        if (M == max) {                                 if necessary, reallocate h
            max += dmax;
            h = (double *) realloc((char *) h, (max + 1) * sizeof(double));
            }
        if (fscanf(fph, "%lf", h + M) == EOF) break;
        }

    M--;                                                                M is filter order

    if (L < M) {
        fprintf(stderr, "blocksize L must be at least %d\n", M);
        exit(0);
        }

    h = (double *) realloc((char *) h, (M + 1) * sizeof(double));       final allocation
    x = (double *) calloc(L, sizeof(double));                           allocate input block
    y = (double *) calloc(L + M, sizeof(double));                       allocate output block
    ytemp = (double *) calloc(M, sizeof(double));                       initialized to zero

                                                                        start filtering:
    for (;;) {                                                          read input block
        for (N=0; N<L; N++)                                             N=L except for last
            if (scanf("%lf", x+N) == EOF) goto last;

        blockcon(M, h, L, x, y, ytemp);                                 process block

        for (i=0; i<L; i++)                                             write output block
            printf("%lf\n", y[i]);
        }

last:                                                                   process last block

           blockcon(M, h, N, x, y, ytemp);                              last block has N<=L

        for (i=0; i<N+M; i++)                                           write entire y block
            printf("%lf\n", y[i]);
}
```


The dimension of the filter array **h** is determined on the fly by reading the file of filter coefficients. Because the length of this file is not known in advance, the program keeps reallocating the length of **h** in increments of MAX coefficients until the final length is determined and the final reallocation is made. The filtering portion of the program is identical to the program segment discussed at the end of Section 4.1.10.

The program can receive its input from `stdin` with the user entering the input samples one at a time. However, only after a group of L samples has been entered followed by a <CR> (carriage return), will the program output the first group of L outputs. Then, the next group of L input samples is entered and the program outputs the corresponding group of L outputs, and so on. At the end, one must enter a <CTRL-Z> (for MSDOS) in order for the program to process the input-off transients.

Problem 4.10

A complete such program is included below.

```
/* firfilt.c - FIR filtering by sample-by-sample processing */

#include <stdlib.h>
#include <stdio.h>

#define MAX 64                                              initial allocation size

double fir();

void main(int argc, char **argv)
{
    FILE *fph;                                              filter file
    double *h, *w, x, y;                                    filter, states, input, output samples
    int M, i;
    int max = MAX, dmax = MAX;                              allocation for h and increment

    if (argc != 2) {
        fprintf(stderr, "usage: firfilt hfile <xfile >yfile\n");
        exit(0);
        }

    if ((fph = fopen(argv[1], "r")) == NULL) {
           fprintf(stderr, "can't open filter file: %s\n", argv[1]);
           exit(0);
           }

    h = (double *) calloc(max + 1, sizeof(double));         preliminary allocation

    for (M=0;; M++) {                                       read h
        if (M == max) {                                     reallocate h, if necessary
            max += dmax;
            h = (double *) realloc((char *) h, (max + 1) * sizeof(double));
            }
        if (fscanf(fph, "%lf", h + M) == EOF) break;
```

```
        }

    M--;                                                    M is filter order

    h = (double *) realloc((char *) h, (M + 1) * sizeof(double));   final allocation
    w = (double *) calloc(M + 1, sizeof(double));

                                                            start filtering:
    while(scanf("%lf", &x) != EOF) {                        keep reading input samples
        y = fir(M, h, w, x);                                compute output sample
        printf("%lf\n", y);                                 write output sample
        }

    for (i=0; i<M; i++) {                                   input-off transients
        y = fir(M, h, w, 0.0);
        printf("%lf\n", y);
        }
}
```

As in the case of `blkfilt.c`, the dimension of the filter array **h** is determined on the fly by reading the file of filter coefficients. The program keeps reallocating the length of **h** in increments of `MAX` coefficients until the final length is determined and the final reallocation is made. The filtering portion of the program is identical to the program segment discussed in Section 4.2.3.

The program can receive its input from `stdin` with the user entering the input one sample at a time. After each input is entered followed by a <CR>, the program computes and outputs the corresponding output sample. (One can also enter a few input samples at a time, followed by a <CR>, and the program will produce the corresponding outputs.) At the end, one must enter a <CTRL-Z> (for MSDOS) in order for the program to process the input-off transients.

Problem 4.11

A complete such program is included below.

```
/* cfirfilt.c - FIR filtering using circular delay-line buffer */

#include <stdlib.h>
#include <stdio.h>

#define MAX 64                                              initial allocation size

double cfir();

void main(int argc, char **argv)
{
    FILE *fph;                                              filter file
    double *h, *w, *p;                                      filter, states, circular pointer
    double x, y;                                            input and output samples
    int M, i;
    int max = MAX, dmax = MAX;                              allocation for h and increment

    if (argc != 2) {
        fprintf(stderr, "usage: cfirfilt hfile <xfile >yfile\n");
        exit(0);
        }
```



```
    if ((fph = fopen(argv[1], "r")) == NULL) {
           fprintf(stderr, "can't open filter file: %s\n", argv[1]);
           exit(0);
           }

    h = (double *) calloc(max + 1, sizeof(double));         preliminary allocation

    for (M=0;; M++) {                                       read h
        if (M == max) {                                     reallocate h, if necessary
            max += dmax;
            h = (double *) realloc((char *) h, (max + 1) * sizeof(double));
            }
        if (fscanf(fph, "%lf", h + M) == EOF) break;
        }

    M--;                                                    M is filter order

    h = (double *) realloc((char *) h, (M + 1) * sizeof(double));
    w = (double *) calloc(M + 1, sizeof(double));
    p = w;                                                  initialize p

                                                            start filtering:
    while(scanf("%lf", &x) != EOF) {                        keep reading input samples
        y = cfir(M, h, w, &p, x);                           compute output sample
        printf("%lf\n", y);                                 write output sample
        }

    for (i=0; i<M; i++) {                                   input-off transients
        y = cfir(M, h, w, &p, 0);
        printf("%lf\n", y);
        }
}
```

It is identical to `firfilt.c`, except it uses the circular version `cfir` instead of `fir` to do the filtering. In addition, the circular pointer *p* is defined and initialized by the program. This program must be compiled and linked with `cfir.c` and `wrap.c`. The version using the routine `cfir1` is identical.

The following version is based on the routine `cfir2` and uses the circular pointer index *q* instead of the pointer *p*. It must be linked with the routines `cfir2` and `wrap2`.

```
/* cfirfil2.c - FIR filtering using circular delay-line buffer */

#include <stdlib.h>
#include <stdio.h>

#define MAX 64                                              initial allocation size

double cfir2();

void main(int argc, char **argv)
{
    FILE *fph;                                              filter file
    double *h, *w;                                          filter, states, circular pointer
    double x, y;                                            input and output samples
    int M, i, q;                                            circular pointer index
    int max = MAX, dmax = MAX;                              allocation for h and increment
```

```
    if (argc != 2) {
        fprintf(stderr, "usage: cfirfil2 hfile <xfile >yfile\n");
        exit(0);
        }

    if ((fph = fopen(argv[1], "r")) == NULL) {
           fprintf(stderr, "can't open filter file: %s\n", argv[1]);
           exit(0);
           }

    h = (double *) calloc(max + 1, sizeof(double));      preliminary allocation

    for (M=0;; M++) {                                    read h
        if (M == max) {                                  reallocate h, if necessary
            max += dmax;
            h = (double *) realloc((char *) h, (max + 1) * sizeof(double));
            }
        if (fscanf(fph, "%lf", h + M) == EOF) break;
        }

    M--;                                                 M is filter order

    h = (double *) realloc((char *) h, (M + 1) * sizeof(double));
    w = (double *) calloc(M + 1, sizeof(double));
    q = 0;                                               initialize q

                                                         start filtering:
    while(scanf("%lf", &x) != EOF) {                     keep reading input samples
        y = cfir2(M, h, w, &q, x);                       compute output sample
        printf("%lf\n", y);                              write output sample
        }

    for (i=0; i<M; i++) {                                input-off transients
        y = cfir2(M, h, w, &q, 0);
        printf("%lf\n", y);
        }
}
```

Problem 4.12

The following C program is an implementation:

```
/* cdelfilt.c - Implementation of delay using circular buffer */

#include <stdlib.h>
#include <stdio.h>

double cdelay();
double tap();

void main(int argc, char **argv)
{
    double *w, *p;                                   linear buffer & circular pointer
    double x, y;                                     input and output samples
    int D, i, j;

    if (argc != 3) {
        fprintf(stderr, "usage: cdelfilt i D <xfile >yfile\n");
        exit(0);
        }

    i = atoi(argv[1]);
    D = atoi(argv[2]);

    if (i>D) {
        fprintf(stderr, "i must be in the range {0, 1, ..., D}\n");
        exit(0);
        }

    w = (double *) calloc(D + 1, sizeof(double));
    p = w;                                           initialize p

    while(scanf("%lf", &x) != EOF) {                 keep reading input samples
        *p = x;                                      store in circular buffer
        y = tap(D, w, p, i);                         y(n) = x(n-i) = i-th delay
        cdelay(D, w, &p);                            update delay line
        printf("%lf\n", y);                          write output sample
        }

    for (j=0; j<i; j++) {                            input-off transients
        *p = 0;                                      zero input
        y = tap(D, w, p, i);
        cdelay(D, w, &p);
        printf("%lf\n", y);
        }
}
```

Problem 4.13

Consider a 3d order filter with internal states $w_i(n) = x(n-i)$, $i = 0, 1, 2, 3$. They are delayed replicas of each other:

$$\begin{aligned} w_3(n) &= w_2(n-1) \\ w_2(n) &= w_1(n-1) \\ w_1(n) &= w_0(n-1) \\ w_0(n) &= x(n) \end{aligned}$$

Thus, the current states are obtained *from the previous* ones by shifting the previous contents of the delay line and reading $x(n)$ into $w_0(n)$. Once this shift is done, the filter's output is computed by the dot product:

$$y(n) = h_0 w_0(n) + h_1 w_1(n) + h_2 w_2(n) + h_3 w_3(n)$$

The following routine `firalt.c` implements this alternative procedure:

```
/* firalt.c - Alternative version of FIR filter in direct form */

double firalt(M, h, w, x)                            Usage: y=firalt(M, h, w, x);
double *h, *w, x;                                    h=filter, w=state, x=input
int M;                                               M=filter order
{
```

```
        int i;
        double y;                               y=output sample

        for (i=M; i>=1; i--)                    update states from previous call
                w[i] = w[i-1];                  done in reverse order

        w[0] = x;                               read current input sample x

        for (y=0, i=0; i<=M; i++)
                y += h[i] * w[i];               process current output sample

        return y;
}
```

Note that the previous state vector is updated first into its current value, the current input is read into w_0, and only then is the output y computed. Upon exit, the states are left at their current values and not updated. They will be updated by the next call. The vector **w** must be initialized to zero prior to the first call, just like the `fir.c` case.

Problem 4.14

Introduce the internal states:

$$w_0(n) = x(n)$$
$$w_1(n) = x(n-1) = w_0(n-1)$$
$$w_2(n) = x(n-2) = w_1(n-1)$$
$$w_3(n) = x(n-3) = w_2(n-1)$$

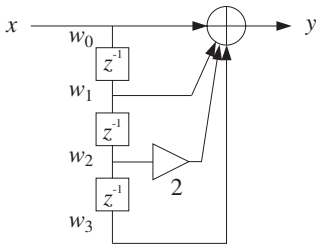
Then, the I/O equation together with the state-updating equations will read:

$$w_0(n) = x(n)$$
$$y(n) = w_0(n) + w_1(n) + 2w_2(n) + w_3(n)$$

and

$$w_3(n+1) = w_2(n)$$
$$w_2(n+1) = w_1(n)$$
$$w_1(n+1) = w_0(n)$$

This leads to the following sample processing algorithm and block diagram:



for each input sample x *do:*
$w_0 = x$
$y = w_0 + w_1 + 2w_2 + w_3$
$w_3 = w_2$
$w_2 = w_1$
$w_1 = w_0$

The sample processing algorithm generates the following output samples:

n	x	w_0	w_1	w_2	w_3	$y = w_0 + w_1 + 2w_2 + w_3$
0	1	1	0	0	0	1
1	2	2	1	0	0	3
2	1	1	2	1	0	5
3	1	1	1	2	1	7
4	2	2	1	1	2	7
5	1	1	2	1	1	6
6	1	1	1	2	1	7
7	1	1	1	1	2	6
8	0	0	1	1	1	4
9	0	0	0	1	1	3
10	0	0	0	0	1	1

The first three outputs correspond to the input-on transients (the internal delay registers are still filling up). The period $3 \le n \le 7$ corresponds to steady state (all delays are filled). The last three outputs, in general the last M outputs for an M-th order FIR filter, are the input-off ($x = 0$) transients (the delays gradually empty out their contents). A C routine and main program implementing this algorithm are given below:

```
#include <stdio.h>
#include <malloc.h>                             declares calloc()

double x[8] = {1,2,1,1,2,1,1,1};

double filter2();

void main()
{
    int n;
    double y, *w;

    w = (double *) calloc(4, sizeof(double));   allocates & initializes w to zero

    for (n=0; n<8; n++) {                       input-on transients & steady-state
           y = filter2(x[n], w);
           printf("%lf\n", y);
           }

    for (n=8; n<11; n++) {                      input-off transients
           y = filter2(0.0, w);
           printf("%lf\n", y);
           }
}

double filter2(x, w)                            usage: y = filter2(x, w);
double x, *w;
{
    double y;

    w[0] = x;                                   read input sample

    y = w[0] + w[1] + 2 * w[2] + w[3];          compute output sample

    w[3] = w[2];                                update internal states
    w[2] = w[1];
    w[1] = w[0];
```

```
    return y;
}
```

Problem 4.15

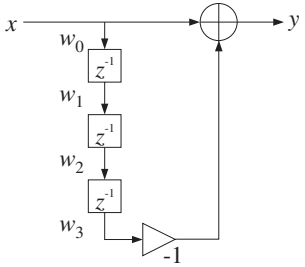
We define the state-variables $w_i(n)$, $i = 0, 1, 2, 3$ as follows:

$$\begin{aligned} w_0(n) &= x(n) \\ w_1(n) &= x(n-1) = w_0(n-1) \\ w_2(n) &= x(n-2) = w_1(n-1) \\ w_3(n) &= x(n-3) = w_2(n-1) \end{aligned}$$

Then, the I/O equation $y(n) = x(n) - x(n-3)$ together with the state-updating equations will read:

$$\begin{aligned} w_0(n) &= x(n) \\ y(n) &= w_0(n) - w_3(n) \end{aligned} \quad \text{and} \quad \begin{aligned} w_3(n+1) &= w_2(n) \\ w_2(n+1) &= w_1(n) \\ w_1(n+1) &= w_0(n) \end{aligned}$$

Note that the output $y(n)$ at time n and the *next* internal states at time $n+1$ are computable from the knowledge of the present input $x(n)$ and the present states at time n. This leads to the following sample processing algorithm and block diagram realization:



for each input sample x do:
$w_0 = x$
$y = w_0 - w_3$
$w_3 = w_2$
$w_2 = w_1$
$w_1 = w_0$

The sample processing algorithm can be applied to the given input samples to generate the output samples:

n	x	w_0	w_1	w_2	w_3	$y = w_0 - w_3$
0	1	1	0	0	0	1
1	1	1	1	0	0	1
2	2	2	1	1	0	2
3	2	2	2	1	1	1
4	4	4	2	2	1	3

Problem 4.16

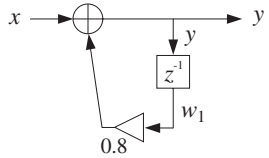
We define the internal state:

$$w_1(n) = y(n-1)$$

Then, the difference equation $y(n) = 0.8y(n-1) + x(n)$ and the state updating equation will read:

$$y(n) = 0.8w_1(n) + x(n) \quad \text{and} \quad w_1(n+1) = y(n)$$

This leads to the following sample processing algorithm and block diagram realization:



for each input sample x do:
$y = 0.8w_1 + x$
$w_1 = y$

The table of computations is:

n	x	w_1	$y = 0.8w_1 + x$
0	1	0.0000	1.0000
1	1	1.0000	1.8000
2	2	1.8000	3.4400
3	2	3.4400	4.7520
4	4	4.7520	7.8016

Problem 4.17

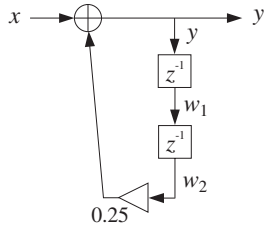
We define the internal states:

$$\begin{aligned} w_1(n) &= y(n-1) \\ w_2(n) &= y(n-2) = w_1(n-1) \end{aligned} \quad \Rightarrow \quad \begin{aligned} w_1(n+1) &= y(n) \\ w_2(n+1) &= y(n-1) = w_1(n) \end{aligned}$$

The difference equation $y(n) = 0.25y(n-2) + x(n)$ and state updating equations become:

$$y(n) = 0.25w_2(n) + x(n) \quad \text{and} \quad \begin{aligned} w_2(n+1) &= w_1(n) \\ w_1(n+1) &= y(n) \end{aligned}$$

This leads to the following sample processing algorithm and block diagram realization:



for each input sample x do:
$y = 0.25w_2 + x$
$w_2 = w_1$
$w_1 = y$

General versions of such routines will be considered in Chapter 7. The sample processing algorithm can be cranked on the given input samples to generate the output:

n	x	$y = 0.25w_2 + x$	w_1	w_2
0	1	1	0	0
1	1	1	1	0
2	2	2.25	1	1
3	2	2.25	2.25	1
4	4	4.5625	2.25	2.25

Notice how the computed y becomes the next w_1 and w_1 shifts into w_2.

Problem 4.18

The impulse response is read off from the coefficients of the I/O equation:

$$\mathbf{h} = [1, 0, -1, 2]$$

The LTI form is implemented by the following table:

n	0	1	2	3	4	5	6	7	8	9	10	
x\h	1	0	-1	2								partial output
1	1	0	-1	2								$x(0)h(n-0)$
1		1	0	-1	2							$x(1)h(n-1)$
2			2	0	-2	4						$x(2)h(n-2)$
2				2	0	-2	4					$x(3)h(n-3)$
2					2	0	-2	4				$x(4)h(n-4)$
2						2	0	-2	4			$x(5)h(n-5)$
1							1	0	-1	2		$x(6)h(n-6)$
1								1	0	-1	2	$x(7)h(n-7)$
$y(n)$	1	1	1	3	2	4	3	3	3	1	2	$\sum_m x(m)h(n-m)$

For the overlap-add method, the input is divided into the following length-4 contiguous blocks:

$$\mathbf{x} = [\underbrace{1, 1, 2, 2}_{\mathbf{x}_0}, \underbrace{2, 2, 1, 1}_{\mathbf{x}_1},]$$

Convolving each block separately with **h** gives:

$$\mathbf{y}_0 = \mathbf{h} * \mathbf{x}_0 = [1, 1, 1, 3, 0, 2, 4]$$

$$\mathbf{y}_1 = \mathbf{h} * \mathbf{x}_1 = [2, 2, -1, 3, 3, 1, 2]$$

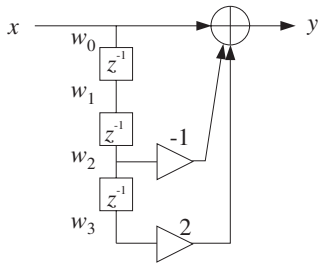
These convolutions can be done by separately folding the two convolution subtables:

	block 0				block 1			
h\x	1	1	2	2	2	2	1	1
1	1	2	2	2	2	2	1	1
0	0	0	0	0	0	0	0	0
-1	-1	-1	-2	-2	-2	-2	-1	-1
2	2	2	4	4	4	4	2	2

The two subblocks begin at the absolute times $n = 0, 4$, respectively. It follows from time-invariance that the corresponding output blocks will also begin at the same absolute times. Thus, aligning the output blocks according to their absolute timings and adding them up gives the final result:

n	0	1	2	3	4	5	6	7	8	9	10
$\mathbf{y}_0$	1	1	1	3	0	2	4				
$\mathbf{y}_1$					2	2	-1	3	3	1	2
$\mathbf{y}$	1	1	1	3	2	4	3	3	3	1	2

The block diagram and sample processing algorithm are:



for each input sample x do:
$w_0 = x$
$y = w_0 - w_2 + 2w_3$
$w_3 = w_2$
$w_2 = w_1$
$w_1 = w_0$

The corresponding circular-buffer version will be:

for each input sample x do:
$s_0 = *p = x$
$s_2 = \text{tap}(3, \mathbf{w}, p, 2)$
$s_3 = \text{tap}(3, \mathbf{w}, p, 3)$
$y = s_0 - s_2 + 2s_3$
cdelay$(3, \mathbf{w}, \&p)$

where tap-1 is not needed, but tap-2 and tap-3 are. Tap-0 s_0 is the content of the current location pointed to by p, which receives the input sample x. After y is computed, the call to `cdelay` circularly decrements the pointer p.

Problem 4.19

Figure P4.2 shows the contents of the registers $\mathbf{w} = [w_0, w_1, w_1]$ at successive time instants. Each time the circular pointer $p = p_{\text{in}}$ points to the w-register that receives the current input, whereas the pointer p_{out} points to the w-register containing the current output, that is, $y(n) = x(n-2)$. Therefore, p_{out} is shifted clockwise by 2 with respect to p_{in}:

$$p_{\text{out}} = p_{\text{in}} + 2 \quad \text{(modulo 3)}$$

As p_{in} gets decremented circularly, so does p_{out}. If p_{in} points to $w[q_{\text{in}}]$ and p_{out} to $w[q_{\text{out}}]$, then

$$p_{\text{in}} = w + q_{\text{in}}$$

$$p_{\text{out}} = w + q_{\text{out}}$$

where

$$q_{\text{out}} = (q_{\text{in}} + 2)\%3$$

Thus, the input and output circular indices take on the sequence of values:

$$q_{\text{in}} = 0, 2, 1, 0, 2, 1, 0, 2, 1, 0, 2$$

$$q_{\text{out}} = 2, 1, 0, 2, 1, 0, 2, 1, 0, 2, 1$$

The following table lists the contents of **w** for both the linear and circular buffer implementations. In the linear case, w_0, w_1 and w_2 are simply the input signal delayed by $i = 0, 1, 2$ units.

In the circular case, the down-arrow indicates the w-register receiving the current input, that is, $w[q_{\text{in}}]$, whereas the up-arrow indicates the corresponding output sample, that is, $w[q_{\text{out}}]$. In the circular case, only one w-register changes value at each time instant.

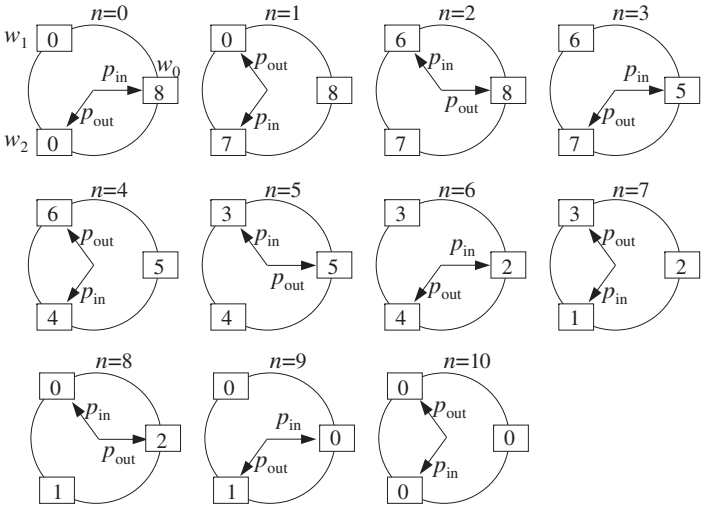


Fig. P4.2 Circular delay-line buffer contents.

		linear			circular					
n	x	w_0	w_1	w_2	w_0	w_1	w_2	q_{in}	q_{out}	y
0	8	8	0	0	8↓	0	0↑	0	2	0
1	7	7	8	0	8	0↑	7↓	2	1	0
2	6	6	7	8	8↑	6↓	7	1	0	8
3	5	5	6	7	5↓	6	7↑	0	2	7
4	4	4	5	6	5	6↑	4↓	2	1	6
5	3	3	4	5	5↑	3↓	4	1	0	5
6	2	2	3	4	2↓	3	4↑	0	2	4
7	1	1	2	3	2	3↑	1↓	2	1	3
8	0	0	1	2	2↑	0↓	1	1	0	2
9	0	0	0	1	0↓	0	1↑	0	2	1
10	0	0	0	0	0	0↑	0↓	2	1	0

Problem 4.20

The transposed form is obtained by the following rules:

1. Exchange input with output.
2. Reverse all signal flows.
4. Replace all adders by nodes.
3. Replace all nodes by adders.

For example, in Fig. 4.2.7, the output adder will be replaced by a node with the signals flowing in reversed directions, and the nodes at the inputs of the delays will be replaced by adders, and $x(n)$ will be exchanged with $y(n)$. The resulting block diagram is shown in Fig. P4.3. We can draw it reversed, as in Fig. 4.3.1.

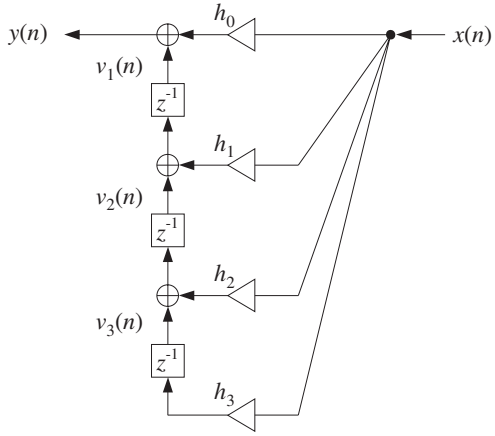


Fig. P4.3 Transposed direct form realization.

Let $v_1(n)$, $v_2(n)$, and $v_3(n)$ be the contents of the three delays at time n. They can be thought of as the *internal states* of the realization. Because of the intervening adders between the delays, the $v_i(n)$ are no longer the delayed versions of each other. Rather, each is a delayed version of its input. These inputs will become the contents of the delays at the *next* time instant. Therefore, the I/O equations describing the filter are in this case:

$$y(n) = v_1(n) + h_0 x(n) \qquad \text{and} \qquad \begin{aligned} v_1(n+1) &= v_2(n) + h_1 x(n) \\ v_2(n+1) &= v_3(n) + h_2 x(n) \\ v_3(n+1) &= h_3 x(n) \end{aligned}$$

This system is equivalent to the original I/O equation (4.2.6). Indeed, down-shifting $v_1(n+1)$, we have

$$v_1(n) = v_2(n-1) + h_1 x(n-1)$$

Therefore,

$$y(n) = h_0 x(n) + v_1(n) = h_0 x(n) + \big(h_1 x(n-1) + v_2(n-1)\big)$$

Similarly, down-shifting $v_2(n+1)$ twice, that is, $v_2(n-1) = v_3(n-2) + h_2 x(n-2)$, and $v_3(n+1)$ three times, that is, $v_3(n-2) = h_3 x(n-3)$, we get

$$\begin{aligned} y(n) &= h_0 x(n) + h_1 x(n-1) + v_2(n-1) \\ &= h_0 x(n) + h_1 x(n-1) + \big(h_2 x(n-2) + v_3(n-2)\big) \\ &= h_0 x(n) + h_1 x(n-1) + h_2 x(n-2) + h_3 x(n-3) \end{aligned}$$

which is equivalent to Eq. (4.2.6). The I/O system leads then to the following sample by sample processing algorithm:

for each input sample x do:
$y = v_1 + h_0 x$
$v_1 = v_2 + h_1 x$
$v_2 = v_3 + h_2 x$
$v_3 = h_3 x$

For Example 4.2.1, Table P4.1 shows the contents of the v_i registers at each time instant. The v_i are initialized to zero. The v_1 and v_2 entries in each row are obtained by forming $v_2 + h_1 x$ and $v_3 + h_2 x$ of the *previous* row. The v_3 column is the delayed (down-shifted) version of the $h_3 x$ column. The final result is the same as in Example 4.2.1.

n	x	$h_0 x$	$h_1 x$	$h_2 x$	$h_3 x$	v_1	v_2	v_3	$y = v_1 + h_0 x$
0	1	1	2	−1	1	0	0	0	1
1	1	1	2	−1	1	2	−1	1	3
2	2	2	4	−2	2	1	0	1	3
3	1	1	2	−1	1	4	−1	2	5
4	2	2	4	−2	2	1	1	1	3
5	2	2	4	−2	2	5	−1	2	7
6	1	1	2	−1	1	3	0	2	4
7	1	1	2	−1	1	2	1	1	3
8	0	0	0	0	0	3	0	1	3
9	0	0	0	0	0	0	1	0	0
10	0	0	0	0	0	1	0	0	1

Table P4.1 Computations in Problem 4.20.

Problem 4.21

The generalization of the transposed algorithm to order M is straightforward:

for each input sample x do:
for $i = 0, 1, \ldots, M-1$ *do:*
$v_i = v_{i+1} + h_i x$
$v_M = h_M x$
$y = v_0$

where for indexing convenience we introduced $v_0(n) = y(n) = v_1(n) + h_0 x(n)$. The following C routine `firtr.c` is an implementation:

```
/* firtr.c - FIR filter in transposed direct form */

double firtr(M, h, v, x)                    Usage: y=firtr(M, h, v, x);
double *h, *v, x;                           h=filter, v=state, x=input
int M;                                      M=filter order
{
       int i;

       for (i=0; i<=M-1; i++)
              v[i] = v[i+1] + h[i] * x;     output & update states

       v[M] = h[M] * x;                     last state

       return v[0];                         y = v0 = v1 + h0x
}
```

The following stand-alone C program `firfiltr.c` is the corresponding version of `firfilt.c`. The only difference is that it uses the routine `firtr.c` instead of `fir.c`:

```
/* firfiltr.c - FIR filtering in transposed form */

#include <stdlib.h>
#include <stdio.h>

#define MAX 64                                  initial allocation size

double firtr();

void main(int argc, char **argv)
{
    FILE *fph;                                  filter file
    double *h, *v, x, y;                        filter array, input, output samples
    int M, i;
    int max = MAX, dmax = MAX;                  allocation for h and increment

    if (argc != 2) {
        fprintf(stderr, "usage: firfilt hfile <xfile >yfile\n");
        exit(0);
        }

    if ((fph = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "can't open filter file: %s\n", argv[1]);
        exit(0);
        }

    h = (double *) calloc(max + 1, sizeof(double));    preliminary allocation

    for (M=0;; M++) {                           read h
        if (M == max) {                         reallocate h, if necessary
             max += dmax;
             h = (double *) realloc((char *) h, (max + 1) * sizeof(double));
             }
        if (fscanf(fph, "%lf", h + M) == EOF) break;
        }

    M--;                                        M is filter order

    h = (double *) realloc((char *) h, (M + 1) * sizeof(double));    final allocation
    v = (double *) calloc((M + 1), sizeof(double));

    while(scanf("%lf", &x) != EOF) {            keep reading input samples
        y = firtr(M, h, v, x);                  compute output sample
        printf("%lf\n", y);                     write output sample
        }

    for (i=0; i<M; i++) {                       input-off transients
```

```
        y = firtr(M, h, v, 0.0);                    compute output sample
        printf("%lf\n", y);
        }
}
```

Transposed forms are convenient in some applications, such as in the implementation of FIR decimators and interpolators in multirate applications. MATLAB uses this form to implement its built-in filtering routines.

Chapter 5 Problems

Problem 5.1

The linearity property follows from:

$$\sum_{n=-\infty}^{\infty}[a_1x_1(n)+a_2x_2(n)]z^{-n} = a_1\sum_{n=-\infty}^{\infty}x_1(n)z^{-n} + a_2\sum_{n=-\infty}^{\infty}x_2(n)z^{-n}$$

The delay property follows from:

$$X_D(z) = \sum_{n=-\infty}^{\infty}x_D(n)z^{-n} = \sum_{n=-\infty}^{\infty}x(n-D)z^{-n} =$$

$$= \sum_{k=-\infty}^{\infty}x(k)z^{-(k+D)} = z^{-D}\sum_{k=-\infty}^{\infty}x(k)z^{-k} = z^{-D}X(z)$$

where we changed summation index from n to $k = n - D$, so that $n = k + D$. The convolution property follows from the linearity and delay properties. Writing $y(n)$ in terms of the delayed replicas of $x(n)$, we have:

$$y(n) = \sum_{m=-\infty}^{\infty}h(m)x(n-m) = \sum_{m=-\infty}^{\infty}h(m)x_m(n)$$

Taking z-transforms of both sides and using the linearity and delay properties, we get:

$$Y(z) = \sum_{m=-\infty}^{\infty}h(m)X_m(z) = \sum_{m=-\infty}^{\infty}h(m)z^{-m}X(z) = \left[\sum_{m=-\infty}^{\infty}h(m)z^{-m}\right]X(z)$$

or,

$$Y(z) = H(z)X(z)$$

Problem 5.2

a. From the delay property of z-transforms, $X(z) = z^{-5}\Delta(z)$, where $\Delta(z) = 1$ is the z-transform of $\delta(n)$. Thus, $X(z) = z^{-5}$. The ROC is the entire z-plane with the exception of the point $z = 0$.

b. Similarly, $X(z) = z^5\Delta(z) = z^5$, but now the ROC must only exclude the point at infinity $z = \infty$.

c. The unit-step has z-transform: $U(z) = 1/(1 - z^{-1})$. Thus,

$$X(z) = z^{-5}U(z) = \frac{z^{-5}}{1 - z^{-1}}$$

with ROC $|z| > 1$.

d. We may work with the definition:

$$X(z) = \sum_{n=-\infty}^{\infty}u(-n+5)z^{-n} = \sum_{n=-\infty}^{5}1\cdot z^{-n}$$

$$= z^{-5} + z^{-4} + z^{-3} + z^{-2} + z^{-1} + 1 + z + z^2 + z^3 + \cdots$$

$$= z^{-5}\left[1 + z + z^2 + z^3 + \cdots\right] = \frac{z^{-5}}{1 - z} = z^{-5}U(z^{-1})$$

The convergence of the series requires the ROC $|z| < 1$.

Alternatively, we recognize that $x(n)$ is the delayed version of $u(-n)$, that is, $x(n) = u(-n+5) = u(-(n-5))$. Using the general property that

$$g(n) \xrightarrow{\mathcal{Z}} G(z) \quad \Rightarrow \quad g(-n) \xrightarrow{\mathcal{Z}} G(z^{-1})$$

we find $X(z) = z^{-5}U(z^{-1})$.

Problem 5.3

a. Using the general result: $a^n u(n) \longrightarrow 1/(1-az^{-1})$, ROC $|z| > |a|$, we obtain: $X(z) = 1/(1-(-0.5)z^{-1}) = 1/(1+0.5z^{-1})$, with ROC $|z| > |-0.5| = 0.5$.

b. *Method 1*: Let $w(n) = u(n) - u(n-10)$ with z-transform $W(z)$. Using the linearity and delay properties, we have

$$W(z) = U(z) - z^{-10}U(z) = (1-z^{-10})U(z) = \frac{1-z^{-10}}{1-z^{-1}}$$

Using the modulation property of z-transforms, we have

$$a^n w(n) \longrightarrow W(z/a) = \frac{1-(z/a)^{-10}}{1-(z/a)^{-1}} = \frac{1-a^{10}z^{-10}}{1-az^{-1}}$$

With $a = -0.5$, we have $X(z) = (1-(-0.5)^{10}z^{-10})/(1+0.5z^{-1})$.
ROC is the whole z-plane minus the point $z = 0$.

Method 2: We recognize that $x(n)$ is the finite sequence

$$x(n) = [1, a, a^2, a^3, a^4, a^5, a^6, a^7, a^8, a^9, 0, 0, 0, \cdots]$$

and use the definition of z-transforms:

$$X(z) = 1 + az^{-1} + a^2z^{-2} + \cdots + a^9z^{-9} = \frac{1-a^{10}z^{-10}}{1-az^{-1}}$$

where in the last step we used the finite geometric series:

$$1 + x + x^2 + \cdots + x^m = \frac{1-x^{m+1}}{1-x}, \qquad \text{with } x = az^{-1} \text{ and } m = 9$$

c. $X(z) = \dfrac{1}{1-0.5z^{-1}} + \dfrac{1}{1+0.5z^{-1}} = \dfrac{2}{(1-0.5z^{-1})(1+0.5z^{-1})} = \dfrac{2}{1-0.25z^{-2}}$, ROC $|z| > 0.5$.

Problem 5.4

Using Euler, $2\cos\theta = e^{j\theta} + e^{-j\theta}$, with $\theta = \pi n/2$ and also the fact that $e^{j\pi/2} = j$ and $e^{-j\pi/2} = -j$, we rewrite

$$\begin{aligned} 2(0.8)^n \cos(\pi n/2) &= (0.8)^n [e^{j\pi n/2} + e^{-j\pi n/2}] \\ &= (0.8)^n [j^n + (-j)^n] = (0.8j)^n + (-0.8j)^n \end{aligned}$$

Thus, the signals in questions (a) and (b) are the same. Their z-transform is

$$X(z) = \frac{1}{1-0.8jz^{-1}} + \frac{1}{1+0.8jz^{-1}} = \frac{2}{(1-0.8jz^{-1})(1+0.8jz^{-1})} = \frac{2}{1+0.64z^{-2}}$$

ROC is $|z| > |0.8j| = 0.8$. An alternative method is to list a few of the signal values and then use the infinite geometric series to sum them up, that is, with $a = 0.8$:

$$x(n) = 2[1, 0, -a^2, 0, a^4, 0, -a^6, 0, a^8, 0, -a^{10}, 0, \cdots]$$

we obtain

$$X(z) = 2[1 - a^2z^{-2} + a^4z^{-4} - a^6z^{-6} + a^8z^{-8} - a^{10}z^{-10} + \cdots] = \frac{2}{1+a^2z^{-2}}$$

where we applied the infinite geometric series

$$1 + x + x^2 + x^3 + \cdots = \frac{1}{1-x}$$

with $x = -a^2z^{-2}$. Convergence of the geometric series requires that $|x| < 1$ or $|-a^2z^{-2}| < 1$ or $|z| > |a|$ or in our case $|z| > 0.8$.

Problem 5.5

The three signals in parts (a,b,c) all have the same z-transform, namely,

$$X(z) = \frac{1}{1-0.25z^{-1}} + \frac{1}{1-4z^{-1}} = \frac{2-4.25z^{-1}}{1-4.25z^{-1}+z^{-2}}$$

They differ only in their region of convergence: In case (a), both terms are causal, therefore $|z| > 0.25$ and $|z| > 4$. Thus, ROC is $|z| > 4$. In case (b), the first term requires $|z| > 0.25$ while the second $|z| < 4$. Combining, we find ROC $0.25 < |z| < 4$. In case (c), both terms are anticausal, thus, $|z| < 0.25$ and $|z| < 4$. Thus, ROC is $|z| < 0.25$. The signal in (d) is unstable from both sides of the time axis. The first term in the z-transform expansion

$$X(z) = -\sum_{n=-\infty}^{-1} (0.25)^n z^{-n} + \sum_{n=0}^{\infty} 4^n z^{-n}$$

would require ROC $|z| < 0.25$ and the second $|z| > 4$. The intersection of the two ROC sets is empty, that is, there is no set of z's for which the z-transform would converge.

Problem 5.6

Method 1: List the values of $x(n)$ and sum the geometric series:

$$\begin{aligned} x(n) &= [1, 0, -1, 0, 1, 0, -1, 0, 1, 0, -1, 0, \ldots] \\ X(z) &= 1 - z^{-2} + z^{-4} - z^{-6} + z^{-8} - \cdots \\ &= 1 + x + x^2 + x^3 + x^4 + \cdots \big|_{x=-z^{-2}} = \frac{1}{1-x}\bigg|_{x=-z^{-2}} \\ &= \frac{1}{1+z^{-2}} \end{aligned}$$

where convergence of the geometric series requires:

$$|x| < 1 \qquad \Rightarrow \qquad |-z^{-2}| < 1 \qquad \Rightarrow \qquad |z| > 1$$

Method 2: Use Euler's formula to split $x(n)$ as the sum of two terms of the form $a^n u(n)$:

$$x(n) = \cos(\pi n/2)\,u(n) = \frac{1}{2}\left[e^{\pi jn/2} + e^{-\pi jn/2}\right]u(n) = \frac{1}{2}\left[j^n + (-j)^n\right]u(n)$$

where we used $e^{\pi j/2} = j$. Taking z-transforms, we find:

$$X(z) = \frac{1/2}{1-jz^{-1}} + \frac{1/2}{1+jz^{-1}} = \frac{1}{(1-jz^{-1})(1+jz^{-1})} = \frac{1}{1+z^{-2}}$$

where we applied the result

$$a^n u(n) \xrightarrow{z} \frac{1}{1-az^{-1}}$$

for $a = j$ and $a = -j$.

Method 3: Recognize that $x(n)$ is periodic with period 4. Defining the length-4 sequence $\mathbf{g} = [1, 0, -1, 0]$, we see that $x(n)$ will be the periodic replication of $g(n)$:

$$\mathbf{x} = [\mathbf{g}, \mathbf{g}, \mathbf{g}, \ldots] \qquad \text{or,}$$

$$x(n) = g(n) + g(n-4) + g(n-8) + g(n-12) + \cdots$$

Taking z-transforms of both sides gives

$$X(z) = G(z) + z^{-4}G(z) + z^{8}G(z) + \cdots = \left(1 + z^{-4} + z^{-8} + z^{-12} + \cdots\right)G(z)$$

Summing up the series and using $G(z) = 1 - z^{-2}$, we get:

$$X(z) = \frac{G(z)}{1-z^{-4}} = \frac{1-z^{-2}}{1-z^{-4}} = \frac{1-z^{-2}}{(1-z^{-2})(1+z^{-2})} = \frac{1}{1+z^{-2}}$$

Problem 5.7

Whenever $X(z)$ is a polynomial, the best method to invert is to just pick out the coefficients of the polynomial. For (a), we have:

$$X(z) = (1-4z^{-2})(1+3z^{-1}) = 1 + 3z^{-1} - 4z^{-2} - 12z^{-3}$$

$$\Rightarrow \quad x(n) = [1, 3, -4, -12]$$

For (b), the signal is non-zero only at $n = -3, 0, 2$:

$$x(n) = 5\delta(n) + 3\delta(n+3) + 2\delta(n-2)$$

Problem 5.8

a. Carry out a partial fraction expansion:

$$X(z) = \frac{3(1+0.3z^{-1})}{1-0.81z^{-2}} = \frac{3(1+0.3z^{-1})}{(1-0.9z^{-1})(1+0.9z^{-1})} = \frac{A}{1-0.9z^{-1}} + \frac{B}{1+0.9z^{-1}}$$

where

$$A = \left[\frac{3(1+0.3z^{-1})}{1+0.9z^{-1}}\right]_{z=0.9} = 2, \qquad B = \left[\frac{3(1+0.3z^{-1})}{1-0.9z^{-1}}\right]_{z=-0.9} = 1$$

The first ROC is $|z| > |0.9| = |-0.9|$, thus both terms will be inverted causally:

$$x(n) = A(0.9)^n u(n) + B(-0.9)^n u(n)$$

As expected, the answer is stable because the ROC contains the unit circle. The second ROC is $|z| < |0.9| = |-0.9|$, thus both terms will be inverted anticausally:

$$x(n) = -A(0.9)^n u(-n-1) - B(-0.9)^n u(-n-1)$$

The answer is unstable, because the ROC does not contain the unit circle.

b. Ordinary partial fraction expansion is not valid in this case because the degree of the numerator is the same as the degree of the denominator. However, we may still have an expansion of the form:

$$\begin{aligned} X(z) &= \frac{6-3z^{-1}-2z^{-2}}{1-0.25z^{-2}} = \frac{6-3z^{-1}-2z^{-2}}{(1-0.5z^{-1})(1+0.5z^{-1})} \\ &= A + \frac{B}{1-0.5z^{-1}} + \frac{C}{1+0.5z^{-1}} \end{aligned}$$

where B and C are determined in the usual manner and A is determined by evaluating $X(z)$ at $z = 0$:

$$A = \left[\frac{6-3z^{-1}-2z^{-2}}{1-0.25z^{-2}}\right]_{z=0} = \left[\frac{6z^2-3z-2}{z^2-0.25}\right]_{z=0} = \frac{-2}{-0.25} = 8$$

$$B = \left[\frac{6-3z^{-1}-2z^{-2}}{1+0.5z^{-1}}\right]_{z=0.5} = -4, \qquad C = \left[\frac{6-3z^{-1}-2z^{-2}}{1-0.5z^{-1}}\right]_{z=-0.5} = 2$$

For the first ROC, $|z| > 0.5$, the last two terms will be inverted causally:

$$x(n) = A\delta(n) + B(0.5)^n u(n) + C(-0.5)^n u(n)$$

For the second ROC, $|z| < 0.5$, the two terms will be inverted anticausally:

$$x(n) = A\delta(n) - B(0.5)^n u(-n-1) - C(-0.5)^n u(-n-1)$$

As expected, only the first inverse is stable because its ROC contains the unit circle.

c. The degree of the numerator is *strictly* greater than the degree of the denominator. The simplest approach in such cases is to use the "remove/restore" method, that is, ignore the numerator completely, do a partial fraction on the denominator, get its inverse z-transform, and finally restore the effect of the numerator. To show these steps, write

$$\begin{aligned} X(z) &= \frac{6+z^{-5}}{1-0.64z^{-2}} = (6+z^{-5})\cdot\left[\frac{1}{1-0.64z^{-2}}\right] \\ &\equiv (6+z^{-5})W(z) = 6W(z)+z^{-5}W(z) \end{aligned}$$

In the time-domain, we have then

$$x(n) = 6w(n)+w(n-5)$$

Thus, the problem is reduced to the problem of finding $w(n)$. Doing a partial fraction expansion on $W(z)$, we find

$$W(z) = \frac{1}{1-0.64z^{-2}} = \frac{1}{(1-0.8z^{-1})(1+0.8z^{-1})} = \frac{A}{1-0.8z^{-1}} + \frac{B}{1+0.8z^{-1}}$$

where $A = B = 0.5$. For the ROC $|z| > 0.8$, the two terms are inverted causally:

$$w(n) = A(0.8)^n u(n)+B(-0.8)^n u(n)$$

Inserting in $x(n) = 6w(n)+w(n-5)$, we find

$$\begin{aligned} x(n) &= 6A(0.8)^n u(n)+6B(-0.8)^n u(n)+A(0.8)^{n-5}u(n-5) \\ &\quad + B(-0.8)^{n-5}u(n-5) \end{aligned}$$

Note that the last two terms are active only for $n \geq 5$. For the ROC $|z| < 0.8$, we have the anticausal/unstable answer:

$$w(n) = -A(0.8)^n u(-n-1)-B(-0.8)^n u(-n-1)$$

which gives for $x(n)$:

$$\begin{aligned} x(n) &= -6A(0.8)^n u(-n-1)-6B(-0.8)^n u(-n-1) \\ &\quad - A(0.8)^{n-5}u(-n+4)-B(-0.8)^{n-5}u(-n+4) \end{aligned}$$

The $u(-n+4)$ was obtained as $u(-(n-5)-1)$. Note also that the last two terms are now active for $-n+4 \geq 0$ or $n \leq 4$, that is, $x(n)$ has a slightly causal part extending to the right up to $n = 4$. This happened because the strictly anticausal signal $w(n)$ was delayed (shifted to the right) by 5 time units by the term $w(n-5)$.

d. The minor new feature of this problem is that the poles are complex-valued. When the poles are complex, they come in conjugate pairs. In this case, the corresponding residues are complex conjugates, too. Thus, only half of the residues need be computed:

$$\begin{aligned} X(z) &= \frac{10+z^{-2}}{1+0.25z^{-2}} = \frac{10+z^{-2}}{(1-0.5jz^{-1})(1+0.5jz^{-1})} \\ &= A + \frac{B}{1-0.5jz^{-1}} + \frac{B^*}{1+0.5jz^{-1}} \end{aligned}$$

Again, the A-term is needed because the degrees of numerator and denominator polynomials are equal. We find

$$A = \left[\frac{10+z^{-2}}{10.25z^{-2}}\right]_{z=0} = \left[\frac{10z^2+1}{z^2+0.25}\right]_{z=0} = \frac{1}{0.25} = 4$$

$$B = \left[\frac{10+z^{-2}}{1+0.5jz^{-1}}\right]_{z=0.5j} = 3$$

We only needed to calculate B, and use B^* for the conjugate pole. For the causal case, we have

$$x(n) = A\delta(n)+B(0.5j)^n u(n)+B^*(-0.5j)^n u(n)$$

Now, because the last two terms are complex conjugates of each other, we may use the general identity $2\mathrm{Re}(z) = z + z^*$ to write

$$B(0.5j)^n u(n)+B^*(-0.5j)^n u(n) = 2\mathrm{Re}[B(0.5j)^n u(n)] = 6(0.5)^n \mathrm{Re}[j^n]u(n)$$

But, $\mathrm{Re}[j^n] = \mathrm{Re}[e^{j\pi n/2}] = \cos(\pi n/2)$ Thus,

$$B(0.5j)^n u(n)+B^*(-0.5j)^n u(n) = 6(0.5)^n \cos(\pi n/2)u(n)$$

and the final result is

$$x(n) = 4\delta(n)+6(0.5)^n \cos(\pi n/2)u(n)$$

For the anticausal case, we obtain the anticausal version of the second term, namely,

$$x(n) = 4\delta(n)-6(0.5)^n \cos(\pi n/2)u(-n-1)$$

e. The partial fraction expansion is:

$$X(z) = \frac{6-2z^{-1}-z^{-2}}{(1-z^{-1})(1-0.25z^{-2})} = \frac{4}{1-z^{-1}} + \frac{1}{1-0.5z^{-1}} + \frac{1}{1+0.5z^{-1}}$$

Because ROC is $|z| > 1 > 0.5$, all terms are inverted causally to give:

$$x(n) = 4u(n)+(0.5)^n u(n)+(-0.5)^n u(n)$$

The answer is marginally stable because the pole $z = 1$ is on the unit circle.

f. The PF expansion gives:

$$\begin{aligned} X(z) &= -4 + \frac{1}{1+4z^{-2}} = -4 + \frac{1}{(1-2jz^{-1})(1+2jz^{-1})} \\ &= -4 + \frac{1/2}{1-2jz^{-1}} + \frac{1/2}{1+2jz^{-1}} \end{aligned}$$

The two regions of convergence are:

$$|z| > 2 \qquad \text{and} \qquad |z| < 2$$

They correspond to the causal and anticausal inverses:

$$x_1(n) = -4\delta(n)+0.5(2j)^n u(n)+0.5(-2j)^n u(n)$$
$$x_2(n) = -4\delta(n)-0.5(2j)^n u(-n-1)+0.5(-2j)^n u(-n-1)$$

Using Euler's formula, we may write:

$$0.5\left[(2j)^n+(-2j)^n\right] = 2^n\frac{1}{2}\left[j^n+(-j)^n\right] = 2^n\frac{1}{2}\left[e^{j\pi n/2}+e^{-j\pi n/2}\right]$$
$$= 2^n\cos(\pi n/2)$$

Thus, we can rewrite:

$$x_1(n) = -4\delta(n)+2^n\cos(\pi n/2)u(n)$$
$$x_2(n) = -4\delta(n)-2^n\cos(\pi n/2)u(-n-1)$$

Only $x_2(n)$ is stable because its ROC contains the unit circle, or, because in the second term n is effectively negative and causes it to decay exponentially for large negative n, that is, writing $n=-|n|$, we have:

$$2^n\cos(\pi n/2)u(-n-1) = 2^{-|n|}\cos(\pi n/2)u(-n-1)\to 0 \qquad \text{as} \quad n\to-\infty$$

g. The PF expansion is in this case:

$$X(z) = \frac{4-0.6z^{-1}+0.2z^{-2}}{(1-0.5z^{-1})(1+0.4z^{-1})} = A+\frac{B}{1-0.5z^{-1}}+\frac{C}{1+0.4z^{-1}}$$

where

$$A = \left[\frac{4-0.6z^{-1}+0.2z^{-2}}{(1-0.5z^{-1})(1+0.4z^{-1})}\right]_{z=0} = \frac{0.2}{-0.5\cdot 0.4} = -1$$
$$B = \left[\frac{4-0.6z^{-1}+0.2z^{-2}}{1+0.4z^{-1}}\right]_{z=0.5} = 2$$
$$C = \left[\frac{4-0.6z^{-1}+0.2z^{-2}}{1-0.5z^{-1}}\right]_{z=-0.4} = 3$$

The three ROC's are:

$$|z|>0.5, \qquad 0.5>|z|>0.4, \qquad 0.4>|z|$$

The corresponding inverses are:

$$x_1(n) = A\delta(n)+B(0.5)^n u(n)+C(-0.4)^n u(n)$$
$$x_2(n) = A\delta(n)-B(0.5)^n u(-n-1)+C(-0.4)^n u(n)$$
$$x_3(n) = A\delta(n)-B(0.5)^n u(-n-1)-C(-0.4)^n u(-n-1)$$

Only $x_1(n)$ is stable. Its ROC contains the unit circle. The B-term in $x_2(n)$ and both the B and C terms of $x_3(n)$ diverge exponentially for large negative n.

Problem 5.9

Consider the z-transform pair:

$$x_a(n) = a^n u(n) \qquad \Leftrightarrow \qquad X_a(z) = \frac{1}{1-az^{-1}}$$

Applying the derivative operator $\partial/\partial a$ to the pair, derive the z-transform of the sequence $x(n) = na^n u(n)$.
Noting that

$$\frac{\partial}{\partial a}x_a(n) = na^{n-1}u(n)$$

we get

$$\frac{\partial}{\partial a}X_a(z) = \frac{\partial}{\partial a}\left(\frac{1}{1-az^{-1}}\right) = \frac{z^{-1}}{(1-az^{-1})^2}$$

Thus,

$$na^{n-1}u(n)\xrightarrow{z}\frac{z^{-1}}{(1-az^{-1})^2}$$

Multiplying by a, we get:

$$\boxed{na^n u(n)\xrightarrow{z}\frac{az^{-1}}{(1-az^{-1})^2}}$$

Problem 5.10

We note that each application of D brings down a factor of n:

$$D(a^n) = a\frac{\partial}{\partial a}(a^n) = ana^{n-1} = na^n$$
$$D^2(a^n) = D(D(a^n)) = D(na^n) = nD(a^n) = n(na^n) = n^2a^n$$
$$D^3(a^n) = D(D^2(a^n)) = D(n^2a^n) = n^2D(a^n) = n^2(na^n) = n^3a^n$$
$$\cdots$$
$$D^k(a^n) = n^k a^n$$

Thus, in the z-domain:

$$\boxed{n^k a^n u(n)\xrightarrow{z}\left(a\frac{\partial}{\partial a}\right)^k\left(\frac{1}{1-az^{-1}}\right)}$$

In particular, for $k=1$, we have:

$$\left(a\frac{\partial}{\partial a}\right)\left(\frac{1}{1-az^{-1}}\right) = \frac{az^{-1}}{(1-az^{-1})^2}$$

Therefore,

$$na^n u(n)\xrightarrow{z}\frac{az^{-1}}{(1-az^{-1})^2}$$

which agrees with the previous problem. For $k=2$, we apply D again to get:

$$\left(a\frac{\partial}{\partial a}\right)^2\left(\frac{1}{1-az^{-1}}\right)=\left(a\frac{\partial}{\partial a}\right)\left(\frac{az^{-1}}{(1-az^{-1})^2}\right)$$

$$=az^{-1}\left[\frac{1}{(1-az)^2}+\frac{2az^{-1}}{(1-az^{-1})^3}\right]$$

$$=\frac{az^{-1}(1+az^{-1})}{(1-az^{-1})^3}$$

Therefore,

$$\boxed{n^2a^nu(n)\xrightarrow{z}\frac{az^{-1}(1+az^{-1})}{(1-az^{-1})^3}}$$

Adding the results of the $k=1$ and $k=2$ cases, we also get:

$$\boxed{n(n+1)a^nu(n)\xrightarrow{z}\frac{2az^{-1}}{(1-az^{-1})^3}}$$

For the cases $k=3,4$, we find after some tedious algebra:

$$n^3a^nu(n)\xrightarrow{z}\frac{az^{-1}(1+4az^{-1}+a^2z^{-2})}{(1-az^{-1})^4}$$

$$n^4a^nu(n)\xrightarrow{z}\frac{az^{-1}(1+11az^{-1}+11a^2z^{-2}+a^3z^{-3})}{(1-az^{-1})^5}$$

Problem 5.11

Defining the square pulses

$$g_1(n)=u(n)-u(n-L)$$

$$g_2(n)=g_1(n+L-1)=u(n+L-1)-u(n-1)$$

and using the z-transform

$$u(n)\xrightarrow{z}U(z)=\frac{1}{1-z^{-1}}$$

we find the z-transforms:

$$G_1(z)=U(z)-z^{-L}U(z)=\frac{1-z^{-L}}{1-z^{-1}}$$

$$G_2(z)=U(z)z^{L-1}-z^{-1}U(z)=\frac{1-z^{-L}}{1-z^{-1}}z^{L-1}$$

Therefore, the right-hand-side of the desired z-transform is the product:

$$X(z)=\frac{1}{L}\left[\frac{1-z^{-L}}{1-z^{-1}}\right]^2z^{L-1}=\frac{1}{L}G_1(z)G_2(z)$$

Thus, in the time domain we have the convolution:

$$x(n)=\frac{1}{L}g_1(n)*g_2(n)=\frac{1}{L}\sum_m g_1(m)g_2(n-m)$$

Because $g_1(n)$ is non-zero only over $0\le n\le L-1$ and $g_2(n)$ over $-(L-1)\le n\le 0$, we obtain the restrictions on the convolution summation indices:

$$0\le m\le L-1$$

$$-(L-1)\le n-m\le 0$$

By adding them, we obtain the overall range of the index n of $x(n)$, that is,

$$-(L-1)\le n\le L-1$$

For any n in this range, we obtain the range of the summation in m:

$$0\le m\le L-1$$

$$0\le m-n\le L-1\quad\Rightarrow\quad n\le m\le n+L-1$$

which combine into the single inequality:

$$\max(0,n)\le m\le\min(L-1,n+L-1)$$

The convolution sum becomes then:

$$x(n)=\frac{1}{L}\sum_{m=\max(0,n)}^{\min(L-1,n+L-1)}g_1(m)g_2(n-m),\qquad -(L-1)\le n\le L-1$$

Because over this range, both $g_1(n)$ and $g_2(n)$ are unity, we obtain:

$$x(n)=\frac{1}{L}\sum_{m=\max(0,n)}^{\min(L-1,n+L-1)}1\cdot 1=\frac{1}{L}\big[\max(L-1,n+L-1)-\min(0,n)+1\big]$$

By checking separately the ranges $-(L-1)\le n\le 0$ and $0\le n\le L-1$, it is easily verified that the above expression is equivalent to:

$$x(n)=\frac{1}{L}\big[\max(L-1,n+L-1)-\min(0,n)+1\big]=1-\frac{|n|}{L},\qquad -(L-1)\le n\le L-1$$

At all other n, including the end-points $n=\pm L$, $x(n)$ is zero.

Problem 5.12

Start with the result

$$a^nu(n)\xrightarrow{z}\frac{1}{1-az^{-1}}$$

and apply it for $a=Re^{j\omega_0}$ and $a^*=Re^{-j\omega_0}$:

$$R^ne^{j\omega_0 n}u(n)\xrightarrow{z}\frac{1}{1-Re^{j\omega_0}z^{-1}}$$

$$R^ne^{-j\omega_0 n}u(n)\xrightarrow{z}\frac{1}{1-Re^{-j\omega_0}z^{-1}}$$

The cosine and sine signals are obtained from the above as the following linear combinations:

$$R^n\cos(\omega_0 n)u(n)=R^n\frac{1}{2}\left[e^{j\omega_0 n}u(n)+e^{-j\omega_0 n}u(n)\right]$$

$$R^n\sin(\omega_0 n)u(n)=R^n\frac{1}{2j}\left[e^{j\omega_0 n}u(n)-e^{-j\omega_0 n}u(n)\right]$$

Therefore, the corresponding linear combinations of z-transforms will be:

$$R^n \cos(\omega_0 n)\, u(n) \xrightarrow{z} \frac{1}{2}\left[\frac{1}{1-Re^{j\omega_0}z^{-1}} + \frac{1}{1-Re^{-j\omega_0}z^{-1}}\right]$$

$$R^n \sin(\omega_0 n)\, u(n) \xrightarrow{z} \frac{1}{2j}\left[\frac{1}{1-Re^{j\omega_0}z^{-1}} - \frac{1}{1-Re^{-j\omega_0}z^{-1}}\right]$$

which simplify into:

$$R^n \cos(\omega_0 n)\, u(n) \xrightarrow{z} \frac{1-R\cos\omega_0\, z^{-1}}{1-2R\cos\omega_0\, z^{-1}+R^2z^{-2}}$$

$$R^n \sin(\omega_0 n)\, u(n) \xrightarrow{z} \frac{R\sin\omega_0\, z^{-1}}{1-2R\cos\omega_0\, z^{-1}+R^2z^{-2}}$$

Problem 5.13

Define the z-transform of one period:

$$A(z) = a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}$$

Then, the z-transform of the periodic signal is recognized as the sum of the delayed replicas of $A(z)$, that is,

$$\begin{aligned} X(z) &= a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} + a_0 z^{-4} + a_1 z^{-5} + a_2 z^{-6} + a_3 z^{-7} \\ &\quad + a_0 z^{-8} + a_1 z^{-9} + a_2 z^{-10} + a_3 z^{-11} + \cdots \\ &= (a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}) + z^{-4}(a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}) \\ &\quad + z^{-8}(a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}) + \cdots \\ &= A(z) + z^{-4}A(z) + z^{-8}A(z) + \cdots \\ &= (1 + z^{-4} + z^{-8} + \cdots)A(z) = \frac{A(z)}{1-z^{-4}} \\ &= \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}{1-z^{-4}} \end{aligned}$$

where the convergence of the geometric series requires the ROC $|z| > 1$.

Problem 5.14

Factoring the denominator $1 - z^{-4}$, we obtain the PF expansion:

$$\begin{aligned} X(z) &= \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}{1-z^{-4}} = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}{(1-z^{-1})(1+z)(1-jz^{-1})(1+jz^{-1})} \\ &= \frac{A}{1-z^{-1}} + \frac{B}{1+z^{-1}} + \frac{C}{1-jz^{-1}} + \frac{C^*}{1+jz^{-1}} \end{aligned}$$

where the coefficients C and C^* are conjugates because they correspond to the conjugate poles $\pm j$ (this follows from the fact that the denominator has real coefficients.)

The PFE coefficients are determined as follows:

$$A = \left[\frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}{(1+z)(1-jz^{-1})(1+jz^{-1})}\right]_{z=1} = \frac{a_0 + a_1 + a_2 + a_3}{4}$$

$$B = \left[\frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}{(1-z^{-1})(1-jz^{-1})(1+jz^{-1})}\right]_{z=-1} = \frac{a_0 - a_1 + a_2 - a_3}{4}$$

$$C = \left[\frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}}{(1-z^{-1})(1+z)(1+jz^{-1})}\right]_{z=j} = \frac{(a_0 - a_2) - j(a_1 - a_3)}{4} \equiv C_R - jC_I$$

Taking (causal) inverse z-transforms of the PF expansion, we obtain for $n \geq 0$:

$$\begin{aligned} x(n) &= A + B(-1)^n + Cj^n + C^*(-j)^n = A + B(-1)^n + Ce^{j\pi n/2} + C^* e^{-j\pi n/2} \\ &= A + B(-1)^n + 2\mathrm{Re}\left[Ce^{j\pi n/2}\right] = A + B(-1)^n + 2\mathrm{Re}\left[(C_R - jC_I)e^{j\pi n/2}\right] \\ &= A + B(-1)^n + 2C_R\cos(\pi n/2) + 2C_I\sin(\pi n/2) \end{aligned}$$

So that finally:

$$\begin{aligned} x(n) &= \frac{a_0 + a_1 + a_2 + a_3}{4} + \frac{a_0 - a_1 + a_2 - a_3}{4}(-1)^n \\ &\quad + \frac{a_0 - a_2}{2}\cos(\pi n/2) + \frac{a_1 - a_3}{2}\sin(\pi n/2) \end{aligned}$$

Setting $n = 0, 1, 2, 3$, we can verify the first four values $\{a_0, a_1, a_2, a_3\}$.

Problem 5.15

Summing the geometric series, we have:

$$X(z) = 1 - z^{-2} + z^{-4} - z^{-6} + z^{-8} - \cdots = \frac{1}{1+z^{-2}}$$

Alternatively, we have:

$$X(z) = 1 - z^{-2} + z^{-4} - z^{-6} + z^{-8} - \cdots = 1 - z^{-2}(1 - z^{-2} + z^{-4} - z^{-6} + z^{-8} - \cdots) = 1 - z^{-2}X(z)$$

which gives

$$(1+z^{-2})X(z) = 1 \qquad \Rightarrow \qquad X(z) = \frac{1}{1+z^{-2}}$$

The inverse z-transform is causal and can be obtained from the PFE:

$$X(z) = \frac{1}{1+z^{-2}} = \frac{0.5}{1-jz^{-1}} + \frac{0.5}{1+jz^{-1}}$$

which gives:

$$x(n) = \left[0.5j^n + 0.5(-j)^n\right]u(n) = \cos(\pi n/2)\, u(n)$$

Problem 5.16

In all cases, we expand in the appropriate geometric series and pick out the coefficients of the expansion.

a. The z-transform and its inverse are:

$$X(z) = \frac{1}{1+z^{-4}} = 1 - z^{-4} + z^{-8} - z^{-12} + z^{-16} - \cdots$$

$$x(n) = [1,0,0,0,-1,0,0,0,1,0,0,0,-1,0,0,0,1,\ldots]$$

b. The z-transform and its inverse are:

$$X(z) = \frac{1}{1-z^{-4}} = 1 + z^{-4} + z^{-8} + z^{-12} + z^{-16} + \cdots$$

$$x(n) = [1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,\ldots]$$

c. The z-transform and its inverse are:

$$X(z) = \frac{1}{1+z^{-8}} = 1 - z^{-8} + z^{-16} - z^{-24} + \cdots$$

$$x(n) = [1,0,0,0,0,0,0,0,-1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,-1,\ldots]$$

d. The z-transform and its inverse are:

$$X(z) = \frac{1}{1-z^{-8}} = 1 + z^{-8} + z^{-16} + z^{-24} + \cdots$$

$$x(n) = [1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,\ldots]$$

Problem 5.17

For case (a), the roots of the denominator are the solutions of $1 + z^{-4} = 0$ or, $z^4 = -1$. They are obtained as follows:

$$z^4 = -1 = e^{j\pi} = e^{j\pi + 2j\pi k} \quad \Rightarrow \quad z = e^{(j\pi+2j\pi k)/4}, \quad k = 0,1,2,3$$

that is, the four numbers

$$z = e^{j\pi/4}, \quad e^{3j\pi/4}, \quad e^{5j\pi/4}, \quad e^{7j\pi/4}$$

which are the conjugate pairs:

$$z = e^{\pm j\pi/4}, \quad e^{\pm 3j\pi/4}$$

Thus, the z-transform factors as follows and has the PF expansion:

$$X(z) = \frac{1}{1+z^{-3}} = \frac{1}{(1-e^{j\pi/4}z^{-1})(1-e^{-j\pi/4}z^{-1})(1-e^{3j\pi/4}z^{-1})(1-e^{-3j\pi/4}z^{-1})}$$

$$= \frac{A}{1-e^{j\pi/4}z^{-1}} + \frac{A^*}{1-e^{-j\pi/4}z^{-1}} + \frac{B}{1-e^{3j\pi/4}z^{-1}} + \frac{B^*}{1-e^{-3j\pi/4}z^{-1}}$$

where the coefficients are:

$$A = \left[\frac{1}{(1-e^{-j\pi/4}z^{-1})(1-e^{3j\pi/4}z^{-1})(1-e^{-3j\pi/4}z^{-1})}\right]_{z=e^{\pi j/4}} = \frac{1}{4}$$

$$B = \left[\frac{1}{(1-e^{j\pi/4}z^{-1})(1-e^{-j\pi/4}z^{-1})(1-e^{-3j\pi/4}z^{-1})}\right]_{z=e^{3\pi j/4}} = \frac{1}{4}$$

Thus, we find the causal inverse for $n \geq 0$:

$$x(n) = \frac{1}{4}e^{j\pi n/4} + \frac{1}{4}e^{-j\pi n/4} + \frac{1}{4}e^{3j\pi n/4} + \frac{1}{4}e^{-j\pi n/4}$$

$$= \frac{1}{2}\cos(\pi n/4) + \frac{1}{2}\cos(3\pi n/4)$$

Cases (b,c,d) are done in a similar fashion. See the solution of Problem 5.19 for the denominator zeros.

Problem 5.18

Let $Q(z)$ and $R(z)$ be the quotient and remainder polynomials of the division of $N(z)$ by $D(z)$. Then, $Q(z)$ will have degree $L-M$, $R(z)$ degree $M-1$. Thus,

$$N(z) = Q(z)D(z) + R(z)$$

It follows that:

$$H(z) = \frac{N(z)}{D(z)} = \frac{Q(z)D(z)+R(z)}{D(z)} = Q(z) + \frac{R(z)}{D(z)}$$

Because the degree of $R(z)$ is strictly less than the degree of $D(z)$, the term $R(z)/D(z)$ will admit a PF expansion of the form:

$$\frac{R(z)}{D(z)} = \sum_{i=1}^{M} \frac{A_i}{1-p_i z^{-1}}$$

where p_i are the M zeros of $D(z)$ or the poles of $H(z)$. Any truly complex poles will come in conjugate pairs, that is, of the type:

$$\frac{A_i}{1-p_i z^{-1}} + \frac{A_i^*}{1-p_i^* z^{-1}}$$

which can be assembled into the 2nd order section:

$$\frac{A_i}{1-p_i z^{-1}} + \frac{A_i^*}{1-p_i^* z^{-1}} = \frac{(A_i + A_i^*) - (A_i p_i^* + A_i^* p_i) z^{-1}}{1 - (p_i + p_i^*) z^{-1} + p_i p_i^* z^{-2}}$$

where the numerator and denominator coefficients are all real.

Problem 5.19

The roots of the polynomial $1 - z^{-D}$ are the solutions of the equation $z^D = 1$, that is, the D-th roots of unity. To find them, replace 1 by $e^{2\pi jk} = 1$, where k is an integer, and take D-th roots of both sides:

$$z^D = 1 = e^{2\pi jk} \qquad \Rightarrow \qquad z_k = e^{j2\pi k/D}, \quad k = 0, 1, \ldots D-1$$

They lie on the unit circle at the angles $\omega_k = 2\pi k/D$, $k = 0, 1, \ldots, D-1$ obtained by dividing the circle into D equal parts. The product of the 1st order zero factors must make up the polynomial $1 - z^{-D}$:

$$\prod_{k=0}^{D-1} (1 - z_k z^{-1}) = 1 - z^{-D}$$

This identity follows from a theorem of algebra that states that if two polynomials have the same set of roots and unity constant coefficient, then they must be identically equal to each other.

For the other case, we have the solutions of $1 + z^{-D} = 0$ or $z^D = -1$. Now, we replace $-1 = e^{\pi j} = e^{j\pi + j2\pi k}$ with any integer k, and find the roots

$$z^D = -1 = e^{j\pi(2k+1)} \qquad \Rightarrow \qquad z_k = e^{j\pi(2k+1)/D}, \qquad k = 0, 1, \ldots, D-1$$

They also lie on the unit circle at D equal divisions, but they are shifted as a whole with respect to the standard D-th roots of unity by a rotation angle of π/D. Fig. P5.1 shows the two cases for $D = 8$.

Fig. P5.1 Roots of $1 - z^{-8}$ and $1 + z^{-8}$.

Problem 5.20

The roots of $1 - az^{-D}$ are the solutions of the equation

$$z^D = a = ae^{j2\pi k} \qquad \Rightarrow \qquad z_k = a^{1/D} e^{j2\pi k/D}, \qquad k = 0, 1, \ldots, D-1$$

They all lie on a circle of radius $a^{1/D}$ at the D-th root of unity angles. Similarly, the roots of $1 + az^{-D}$, lie on the same circle, but at the shifted D-th root of unity angles.

Problem 5.21

Let $y(n) = a^n x(n)$. Its z-transform is:

$$Y(z) = \sum_n y(n) z^{-n} = \sum_n a^n x(n) z^{-n} = \sum_n x(n) (z/a)^{-n}$$

Comparing with

$$X(z) = \sum_n x(n) z^{-n}$$

we see that

$$Y(z) = X(z/a)$$

Since the spectrum $X(\omega)$ is obtained by replacing $z = e^{j\omega}$, the scaled z will be

$$z/a = e^{j\omega}/e^{j\omega_0} = e^{j(\omega - \omega_0)}$$

Thus, the relationship $Y(e^{j\omega}) = X(e^{j(\omega-\omega_0)})$ may be written as

$$Y(\omega) = X(\omega - \omega_0)$$

Alternatively, we may work directly with the definition of the DTFT:

$$Y(\omega) = \sum_n y(n) e^{-j\omega n} = \sum_n e^{j\omega_0 n} x(n) e^{-j\omega n} = \sum_n x(n) e^{-j(\omega-\omega_0)n} = X(\omega - \omega_0)$$

Problem 5.22

We use the identity

$$\frac{1}{2\pi} \int_{-\pi}^{\pi} e^{j\omega(n-m)} d\omega = \delta(n-m)$$

for n, m integers. Indeed, for $n = m$ the left hand side is unity. And, for $n \neq m$, we have:

$$\frac{1}{2\pi} \int_{-\pi}^{\pi} e^{j\omega(n-m)} d\omega = \frac{\cos(\pi(n-m)) - \cos(-\pi(n-m))}{2\pi j(n-m)} = 0$$

Inserting the definition of the DTFT into the inversion formula, we get

$$\begin{aligned} \frac{1}{2\pi} \int_{-\pi}^{\pi} X(\omega) e^{j\omega n} d\omega &= \frac{1}{2\pi} \int_{-\pi}^{\pi} \left(\sum_m x(m) e^{-j\omega m} \right) e^{j\omega n} d\omega \\ &= \sum_m x(m) \frac{1}{2\pi} \int_{-\pi}^{\pi} e^{j\omega(n-m)} d\omega = \sum_m x(m) \delta(n-m) \\ &= x(n) \end{aligned}$$

where we interchanged the series with the integral (this step is justified by assuming that the series is convergent, that is, the unit circle lies in the ROC or that the signal $x(n)$ is strictly stable).

Problem 5.23

We may assume in general that $x(n)$ is complex-valued. Then, we have:

$$\begin{aligned} X(\omega) &= \sum_n x(n) e^{-j\omega n} \\ X^*(\omega) &= \sum_n x^*(n) e^{j\omega n} \end{aligned}$$

Using the inverse DTFT formula, it follows that

$$\frac{1}{2\pi}\int_{-\pi}^{\pi} X(\omega)X^*(\omega)\,d\omega = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(\omega)\left(\sum_n x^*(n)e^{j\omega n}\right)d\omega$$
$$= \sum_n x^*(n)\left(\frac{1}{2\pi}\int_{-\pi}^{\pi} X(\omega)e^{j\omega n}\,d\omega\right)$$
$$= \sum_n x^*(n)x(n)$$

Problem 5.24

For a real-valued signal $x(n)$, we have:

$$X(\omega) = \sum_n x(n)e^{-j\omega n}$$
$$X^*(\omega) = \sum_n x(n)e^{j\omega n}$$
$$X^*(-\omega) = \sum_n x(n)e^{-j\omega n}$$

where in the third equation, we replaced ω by $-\omega$. Comparing the first and third equations, we obtain:

$$X^*(-\omega) = X(\omega)$$

Writing $X(\omega)$ in its polar form, e.g., $X(\omega) = |X(\omega)|e^{j\arg X(\omega)}$, the hermitian property reads:

$$\left(|X(-\omega)|e^{j\arg X(-\omega)}\right)^* = |X(\omega)|e^{j\arg X(\omega)} \qquad \text{or,}$$

$$|X(-\omega)|e^{-j\arg X(-\omega)} = |X(\omega)|e^{j\arg X(\omega)}$$

Equating magnitudes and phases, we get the conditions:

$$|X(-\omega)| = |X(\omega)|, \qquad \arg X(-\omega) = -\arg X(\omega)$$

that is, the magnitude spectrum is even in ω and the phase spectrum is odd.

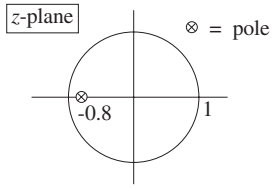
Chapter 6 Problems

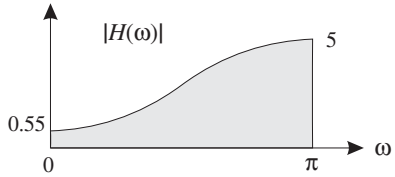
Problem 6.1

a. From $Y(z) = -0.8z^{-1}Y(z) + X(z)$ it follows:

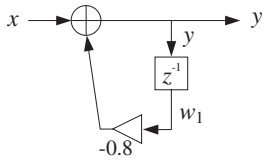
$$H(z) = \frac{1}{1+0.8z^{-1}} \qquad \Rightarrow \qquad H(\omega) = \frac{1}{1+0.8e^{-j\omega}}$$

The causal inverse z-transform of $H(z)$ is $h(n) = (-0.8)^n u(n)$. There is only one pole at $z = -0.8$, that is, near the "high frequency" part of the unit circle. Thus, the filter will tend to enhance high frequencies, i.e., it will behave as a high pass filter:





Block diagram realization and the sample-by-sample processing algorithm:



for each input sample x do:
$y = -0.8w_1 + x$
$w_1 = y$

b. Change -0.8 to 0.8 in the above problem. Now the pole at $z = 0.8$ is in the "low frequency" part of the unit circle and the filter is acting as a low pass filter.

c. The I/O equation $Y(z) = 0.8z^{-1}Y(z) + X(z) + z^{-1}X(z)$ gives

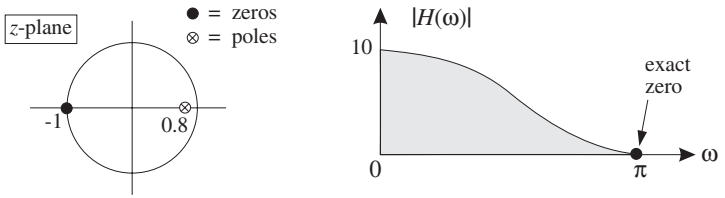
$$H(z) = \frac{1+z^{-1}}{1-0.8z^{-1}} = A + \frac{B}{1-0.8z^{-1}} \quad \Rightarrow \quad h(n) = A\delta(n) + B(0.8)^n u(n)$$

where

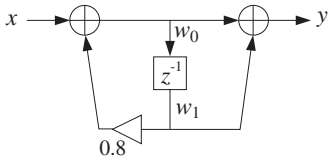
$$A = \left[\frac{1+z^{-1}}{1-0.8z^{-1}}\right]_{z=0} = -1.25, \qquad B = \left[1+z^{-1}\right]_{z=0.8} = 2.25$$

The filter enhances low frequencies for two reasons: first, it has a pole in the low frequency range, $z = 0.8$, and it has a zero in the high frequency range, $z = -1$. Its frequency response will be

$$H(\omega) = \frac{1+e^{-j\omega}}{1-0.8e^{-j\omega}}$$

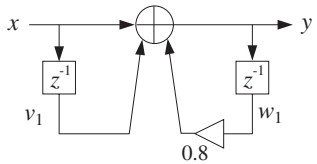


The *canonical form* realization and its sample processing algorithm are:



for each input sample x do:
$w_0 = 0.8w_1 + x$
$y = w_0 + w_1$
$w_1 = w_0$

The *direct form* realization and its sample processing algorithm are:



for each input sample x do:
$y = 0.8w_1 + x + v_1$
$v_1 = x$
$w_1 = y$

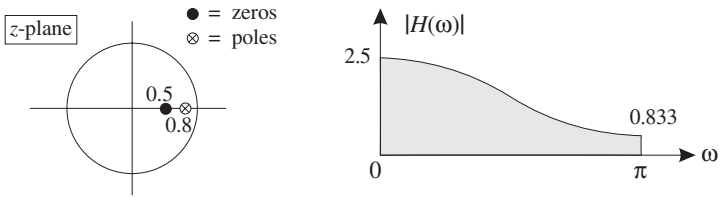
d. The I/O equation $Y(z) = 0.8z^{-1}Y(z) + X(z) - 0.5z^{-1}X(z)$ gives

$$H(z) = \frac{1 - 0.5z^{-1}}{1 - 0.8z^{-1}} = A + \frac{B}{1 - 0.8z^{-1}} \quad \Rightarrow \quad h(n) = A\delta(n) + B(0.8)^n u(n)$$

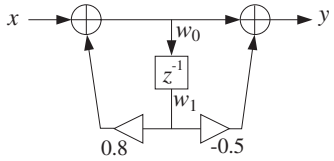
where

$$A = \left[\frac{1 - 0.5z^{-1}}{1 - 0.8z^{-1}}\right]_{z=0} = 0.625, \qquad B = \left[1 - 0.5z^{-1}\right]_{z=0.8} = 0.375$$

The filter has a zero at $z = 0.5$ and a pole at $z = 0.8$ — both in the low frequency range. Thus, their effect will be to cancel each other, and whichever is closest to the unit circle ultimately wins. Here, the pole is nearer. Thus, the filter will tend to act as a lowpass filter. Indeed, its response at $\omega = 0$ or $z = 1$ is $(1 - 0.5)/(1 - 0.8) = 2.5$, whereas its response at $\omega = \pi$ or $z = -1$ is $(1 + 0.5)/(1 + 0.8) = 0.833$.

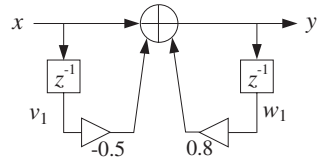


The *canonical form* realization and its sample processing algorithm are:



for each input sample x do:
$w_0 = 0.8w_1 + x$
$y = w_0 - 0.5w_1$
$w_1 = w_0$

The *direct form* realization and its sample processing algorithm are:



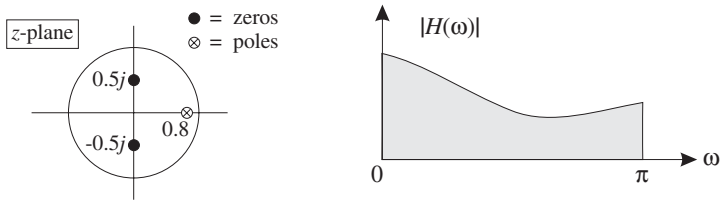
for each input sample x do:
$y = 0.8w_1 + x - 0.5v_1$
$v_1 = x$
$w_1 = y$

e. $Y(z) = 0.8z^{-1}Y(z) + X(z) + 0.25z^{-2}X(z) \quad \Rightarrow \quad H(z) = \dfrac{1 + 0.25z^{-2}}{1 - 0.8z^{-1}}$.

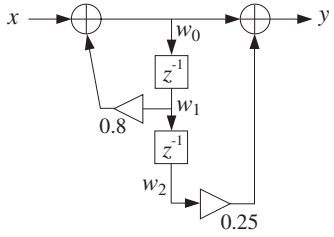
Using the "remove/restore numerator" method of problem (2h), we find

$$h(n) = (0.8)^n u(n) + 0.25(0.8)^{n-2}u(n-2)$$

The filter has two conjugate zeros at midrange, $z = \pm 0.5j = 0.5e^{\pm j\pi/2}$, and a low frequency pole at $z = 0.8$. Thus, the filter will enhance low frequencies and suppress midrange frequencies:

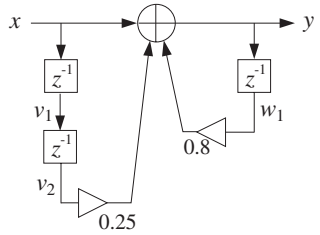


The *canonical form* realization and its sample processing algorithm are:



for each input sample x do:
$w_0 = 0.8w_1 + x$
$y = w_0 + 0.25w_2$
$w_2 = w_1$
$w_1 = w_0$

The *direct form* realization and its sample processing algorithm are:



for each input sample x do:
$y = x + 0.25v_2 + 0.8w_1$
$v_2 = v_1$
$v_1 = x$
$w_1 = y$

f. $Y(z) = 0.9z^{-1}Y(z) - 0.2z^{-2}Y(z) + X(z) + z^{-1}X(z) - 6z^{-2}X(z)$, which implies

$$H(z) = \frac{1 + z^{-1} - 6z^{-2}}{1 - 0.9z^{-1} + 0.2z^{-2}}$$

Factor numerator and denominator and expand in partial fractions:

$$H(z) = \frac{(1+3z^{-1})(1-2z^{-1})}{(1-0.4z^{-1})(1-0.5z^{-1})} = A + \frac{B}{1-0.4z^{-1}} + \frac{C}{1-0.5z^{-1}}$$

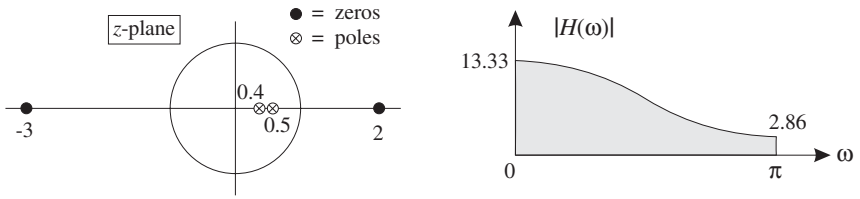
where

$$A = \left[\frac{(1+3z^{-1})(1-2z^{-1})}{(1-0.4z^{-1})(1-0.5z^{-1})}\right]_{z=0} = -30$$

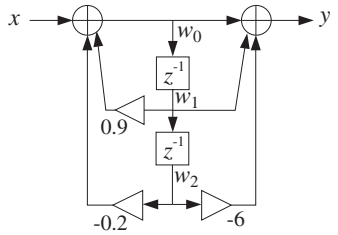
$$B = \left[\frac{(1+3z^{-1})(1-2z^{-1})}{1-0.5z^{-1}}\right]_{z=0.4} = 136$$

$$C = \left[\frac{(1+3z^{-1})(1-2z^{-1})}{1-0.4z^{-1}}\right]_{z=0.5} = -105$$

Thus, $h(n) = A\delta(n) + B(0.4)^n u(n) + C(0.5)^n u(n)$. The two zeros at $z = -3$ and $z = 2$ are too far from the unit circle to have any significant effect on the magnitude response. The two poles at $z = 0.4, 0.5$ are both in the low frequency range. Thus, the filter will be a low pass filter. The value of the magnitude response at $\omega = 0$ or $z = 1$ is $|1+1-6|/|1-0.5||1-0.4| = 13.33$, whereas its value at $\omega = \pi$ or $z = -1$ is $|1-1-6|/|1+0.5||1+0.4| = 2.86$.

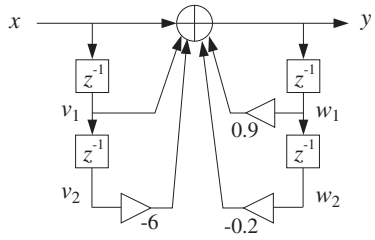


The *canonical form* realization and its sample processing algorithm are:



for each input sample x do:
$w_0 = x + 0.9w_1 - 0.2w_2$
$y = w_0 + w_1 - 6w_2$
$w_2 = w_1$
$w_1 = w_0$

The *direct form* realization and its sample processing algorithm are:



for each input sample x do:
$y = x + v_1 - 6v_2 + 0.9w_1 - 0.2w_2$
$v_2 = v_1$
$v_1 = x$
$w_2 = w_1$
$w_1 = y$

Problem 6.2

a. Setting $x(n) = u(n)$ in the I/O difference equation, we find

$$y(n) = u(n) + 6u(n-1) + 11u(n-2) + 6u(n-3)$$

The 4th term is active only for $n \geq 3$, the 3d term is active only $n \geq 2$, and the 2nd term only for $n \geq 2$. Thus, evaluating at a few n's we find:

$$y(n) = [1, 7, 18, 24, 24, 24, 24, \ldots]$$

The first 3 outputs are the initial transients, the remaining constant values are the steady part. Note that when you send in a unit step, the output always settles to a constant value (for a stable filter). That constant value can be easily precalculated as $H(1)$. In the present case, $H(1) = 1 + 6 + 11 + 6 = 24$. For the alternating step, we have:

$$Y(z) = \frac{1 + 6z^{-1} + 11z^{-2} + 6z^{-3}}{1 + z^{-1}} = 1 + 5z^{-1} + 6z^{-2}$$

$$y(n) = [1, 5, 6, 0, 0, 0, \cdots]$$

There is a pole zero cancellation: the filter has a zero at $z = -1$, cutting off the high-frequency input.

b. Noting that the input z-transform is $X(z) = 1/(1 - z^{-1})$, we find

$$Y(z) = H(z)X(z) = \frac{1 - z^{-4}}{1 - z^{-1}} = 1 + z^{-1} + z^{-2} + z^{-3}$$

where we used the finite geometric series. Inverting, we get

$$y(n) = [1, 1, 1, 1, 0, 0, 0, 0, \dots]$$

The first four samples are the transients, the remaining zeros are the steady state. In the steady-state, the filter cuts off the unit step from going through because the filter has a zero at $z = 1$. The pole of the input canceled the zero of the filter. For the alternating step:

$$Y(z) = \frac{1 - z^{-4}}{1 + z^{-1}} = 1 - z^{-1} + z^{-2} - z^{-3} \quad \Rightarrow \quad y(n) = [1, -1, 1, -1, 0, 0, 0, \cdots]$$

Again, the filter has a high frequency zero at $z = -1$, cutting off the high frequency input.

Problem 6.3

For filter (a), we have:

$$\begin{aligned} Y(z) &= \frac{1}{(1 - z^{-1})(1 - 0.25z^{-2})} = \frac{1}{(1 - z^{-1})(1 - 0.5z^{-1})(1 + 0.5z^{-1})} \\ &= \frac{A}{1 - z^{-1}} + \frac{B}{1 - 0.5z^{-1}} + \frac{C}{1 + 0.5z^{-1}} \\ y(n) &= Au(n) + B(0.5)^n u(n) + C(-0.5)^n u(n) \end{aligned}$$

where $A = H(1) = 4/3$, $B = -0.5$, $C = 1/6$. The A-term is the steady part. For the alternating step, we have:

$$\begin{aligned} Y(z) &= \frac{1}{(1 + z^{-1})(1 - 0.25z^{-2})} = \frac{1}{(1 + z^{-1})(1 - 0.5z^{-1})(1 + 0.5z^{-1})} \\ &= \frac{A}{1 + z^{-1}} + \frac{B}{1 - 0.5z^{-1}} + \frac{C}{1 + 0.5z^{-1}} \\ y(n) &= A(-1)^n u(n) + B(0.5)^n u(n) + C(-0.5)^n u(n) \end{aligned}$$

where $A = H(-1) = 4/3$, $B = 1/6$, $C = -0.5$. The A-term is the steady part. For (b), we have:

$$\begin{aligned} Y(z) &= \frac{1}{(1 - z^{-1})(1 + 0.25z^{-2})} = \frac{1}{(1 - z^{-1})(1 - 0.5jz^{-1})(1 + 0.5jz^{-1})} \\ &= \frac{A}{1 - z^{-1}} + \frac{B}{1 - 0.5jz^{-1}} + \frac{B^*}{1 + 0.5jz^{-1}} \\ y(n) &= Au(n) + B(0.5j)^n u(n) + B^*(-0.5j)^n u(n) \end{aligned}$$

where $A = H(1) = 4/5$ and

$$B = \left[\frac{1}{(1 - z^{-1})(1 + 0.5jz^{-1})} \right]_{z=0.5j} = 0.1 - 0.2j$$

And, for the alternating step:

$$\begin{aligned} Y(z) &= \frac{1}{(1 + z^{-1})(1 + 0.25z^{-2})} = \frac{1}{(1 + z^{-1})(1 - 0.5jz^{-1})(1 + 0.5jz^{-1})} \\ &= \frac{A}{1 + z^{-1}} + \frac{B}{1 - 0.5jz^{-1}} + \frac{B^*}{1 + 0.5jz^{-1}} \\ y(n) &= A(-1)^n u(n) + B(0.5j)^n u(n) + B^*(-0.5j)^n u(n) \end{aligned}$$

where $A = H(-1) = 4/5$ and

$$B = \left[\frac{1}{(1 + z^{-1})(1 + 0.5jz^{-1})} \right]_{z=0.5j} = 0.1 + 0.2j$$

Problem 6.4

For a unit-step input and the filter (a) of Problem 6.1, we have:

$$\begin{aligned} Y(z) &= H(z)X(z) = \frac{1}{(1 + 0.8z^{-1})(1 - z^{-1})} = \frac{A}{1 + 0.8z^{-1}} + \frac{B}{1 - z^{-1}} \\ y(n) &= A(-0.8)^n u(n) + Bu(n) \end{aligned}$$

where $A = 1/2.25$, $B = H(1) = 1/1.8$. The B-term represents the steady part. For filter (b):

$$\begin{aligned} Y(z) &= H(z)X(z) = \frac{1}{(1 - 0.8z^{-1})(1 - z^{-1})} = \frac{A}{1 - 0.8z^{-1}} + \frac{B}{1 - z^{-1}} \\ y(n) &= A(0.8)^n u(n) + Bu(n) \end{aligned}$$

where $A = -4$, $B = H(1) = 5$. The B-term represents the steady part. For filter (c):

$$\begin{aligned} Y(z) &= \frac{1 + z^{-1}}{(1 - z^{-1})(1 - 0.8z^{-1})} = \frac{A}{1 - z^{-1}} + \frac{B}{1 - 0.8z^{-1}} \\ y(n) &= Au(n) + B(0.8)^n u(n) \end{aligned}$$

with $A = H(1) = 10$, $B = -9$. For filter (d):

$$\begin{aligned} Y(z) &= \frac{1 - 0.5z^{-1}}{(1 - z^{-1})(1 - 0.8z^{-1})} = \frac{A}{1 - z^{-1}} + \frac{B}{1 - 0.8z^{-1}} \\ y(n) &= Au(n) + B(0.8)^n u(n) \end{aligned}$$

with $A = H(1) = 2.5$, $B = -1.5$. For filter (e):

$$\begin{aligned} Y(z) &= \frac{1 + 0.25z^{-2}}{(1 - z^{-1})(1 - 0.8z^{-1})} = A + \frac{B}{1 - z^{-1}} + \frac{C}{1 - 0.8z^{-1}} \\ y(n) &= A\delta(n) + Bu(n) + C(0.8)^n u(n) \end{aligned}$$

where $A = Y(0) = 0.3125$, $B = H(1) = 6.25$, $C = -5.5625$. The B-term is the steady part. For filter (f):

$$\begin{aligned} Y(z) &= \frac{1 + z^{-1} - 6z^{-2}}{(1 - z^{-1})(1 - 0.4z^{-1})(1 - 0.5z^{-1})} \\ &= \frac{A}{1 - z^{-1}} + \frac{B}{1 - 0.4z^{-1}} + \frac{C}{1 - 0.5z^{-1}} \\ y(n) &= Au(n) + B(0.4)^n u(n) + C(0.5)^n u(n) \end{aligned}$$

For the alternating step and filter (a) of Problem 6.1, we have:

$$\begin{aligned} Y(z) &= \frac{1}{(1 + 0.8z^{-1})(1 + z^{-1})} = \frac{A}{1 + 0.8z^{-1}} + \frac{B}{1 + z^{-1}} \\ y(n) &= A(-0.8)^n u(n) + B(-1)^n u(n) \end{aligned}$$

where $A = -4$, $B = H(-1) = 5$. The B-term represents the steady part. For filter (b):

$$Y(z) = \frac{1}{(1-0.8z^{-1})(1+z^{-1})} = \frac{A}{1-0.8z^{-1}} + \frac{B}{1+z^{-1}}$$

$$y(n) = A(0.8)^n u(n) + B(-1)^n u(n)$$

where $A = 1/2.25$, $B = H(-1) = 1/1.8$. The B-term represents the steady part. For filter (c):

$$Y(z) = \frac{1+z^{-1}}{(1+z^{-1})(1-0.8z^{-1})} = \frac{1}{1-0.8z^{-1}}$$

$$y(n) = (0.8)^n u(n)$$

The high frequency zero at $z = -1$ of the filter canceled the high frequency input signal. The output is only transient — it decays to zero exponentially. For filter (d):

$$Y(z) = \frac{1-0.5z^{-1}}{(1+z^{-1})(1-0.8z^{-1})} = \frac{A}{1+z^{-1}} + \frac{B}{1-0.8z^{-1}}$$

$$y(n) = A(-1)^n u(n) + B(0.8)^n u(n)$$

For filter (e):

$$Y(z) = \frac{1+0.25z^{-2}}{(1+z^{-1})(1-0.8z^{-1})} = A + \frac{B}{1+z^{-1}} + \frac{C}{1-0.8z^{-1}}$$

$$y(n) = A\delta(n) + B(-1)^n u(n) + C(0.8)^n u(n)$$

The B-term is the steady part. For filter (f):

$$Y(z) = \frac{1+z^{-1}-6z^{-2}}{(1+z^{-1})(1-0.4z^{-1})(1-0.5z^{-1})}$$

$$= \frac{A}{1+z^{-1}} + \frac{B}{1-0.4z^{-1}} + \frac{C}{1-0.5z^{-1}}$$

$$y(n) = A(-1)^n u(n) + B(0.4)^n u(n) + C(0.5)^n u(n)$$

For question (c), the input z-transform is $X(z) = 1/(1-0.5z^{-1})$. Thus, applied to Problem 6.1(d) gives:

$$Y(z) = \frac{1-0.5z^{-1}}{(1-0.5z^{-1})(1-0.8z^{-1})} = \frac{1}{1-0.8z^{-1}}$$

$$y(n) = (0.8)^n u(n)$$

The filter zero canceled the signal pole. For question (d), we have $x(n) = (0.5)^n \cos(\pi n/2)\, u(n) \xrightarrow{z} X(z) = 1/(1+0.25z^{-2})$. Thus,

$$Y(z) = \frac{1+0.25z^{-2}}{(1+0.25z^{-2})(1-0.8z^{-1})} = \frac{1}{1-0.8z^{-1}}$$

$$y(n) = (0.8)^n u(n)$$

Again, the signal poles were canceled by the filter zeros. For question (e), the input has $X(z) = 1/(1-2z^{-1})$. Therefore,

$$Y(z) = \frac{(1+3z^{-1})(1-2z^{-1})}{(1-2z^{-1})(1-0.4z^{-1})(1-0.5z^{-1})}$$

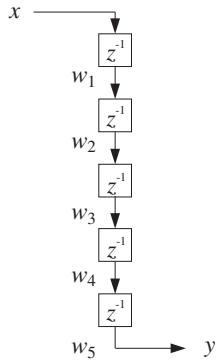
$$= \frac{1+3z^{-1}}{(1-0.4z^{-1})(1-0.5z^{-1})} = \frac{A}{1-0.4z^{-1}} + \frac{B}{1-0.5z^{-1}}$$

$$y(n) = A(0.4)^n u(n) + B(0.5)^n u(n)$$

The unstable input was canceled by a filter zero to give a stable output.

Problem 6.5

Filter (a): $H(z) = z^{-5}$ with I/O equation $Y(z) = H(z)X(z) = z^{-5}X(z)$ and in the time domain $y(n) = x(n-5)$. $H(\omega) = e^{-5j\omega}$ and $|H(\omega)| = 1$, i.e., flat in ω. Block diagram realization and sample-by-sample processing algorithm:



for each input sample x do:
$y = w_5$
$w_5 = w_4$
$w_4 = w_3$
$w_3 = w_2$
$w_2 = w_1$
$w_1 = x$

Filter (b): $H(z) = z^{-5}U(z) = z^{-5}/(1-z^{-1})$ and the I/O equation gives

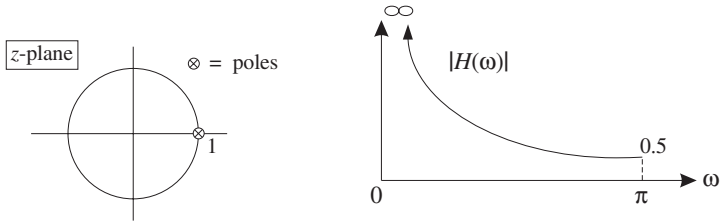
$$Y(z) = H(z)X(z) = \frac{z^{-5}X(z)}{1-z^{-1}} \quad \Rightarrow \quad (1-z^{-1})Y(z) = z^{-5}X(z) \quad \Rightarrow$$

$$Y(z) = z^{-1}Y(z) + z^{-5}X(z) \quad \Rightarrow \quad y(n) = y(n-1) + x(n-5)$$

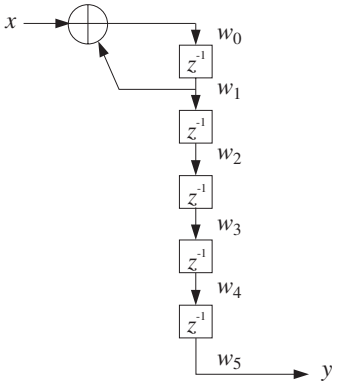
Its frequency response is

$$H(\omega) = \frac{e^{-5j\omega}}{1-e^{-j\omega}} \quad \Rightarrow \quad |H(\omega)| = \frac{1}{2|\sin(\omega/2)|}$$

The pole at $z = 1$ or $\omega = 0$ dominates the frequency response. The filter acts as a lowpass filter (it is actually an accumulator with delay).

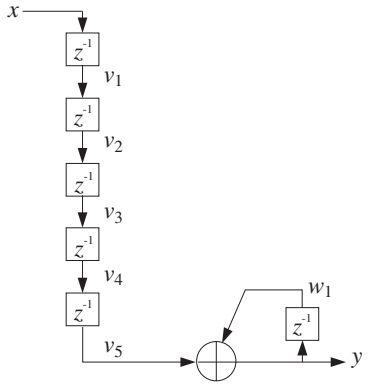


The *canonical form* realization and its sample processing algorithm are:



for each input sample x do:
$w_0 = x + w_1$
$y = w_5$
$w_5 = w_4$
$w_4 = w_3$
$w_3 = w_2$
$w_2 = w_1$
$w_1 = w_0$

The *direct form* realization and its sample processing algorithm are:



for each input sample x do:
$y = w_1 + v_5$
$w_1 = y$
$v_5 = v_4$
$v_4 = v_3$
$v_3 = v_2$
$v_2 = v_1$
$v_1 = x$

Filter (c): $H(z) = 1/(1 - 0.8z^{-1})$ and the I/O equation becomes:

$$Y(z) = H(z)X(z) = \frac{X(z)}{1 - 0.8z^{-1}} \quad \Rightarrow \quad (1 - 0.8z^{-1})Y(z) = X(z) \quad \Rightarrow$$

$$Y(z) = 0.8z^{-1}Y(z) + X(z)$$

and in the time domain

$$y(n) = 0.8y(n-1) + x(n)$$

The rest is as in problem (3d). Filter (d): $H(z) = 1/(1 + 0.8z^{-1})$ and the I/O equation becomes:

$$Y(z) = H(z)X(z) = \frac{X(z)}{1 + 0.8z^{-1}} \quad \Rightarrow \quad (1 + 0.8z^{-1})Y(z) = X(z) \quad \Rightarrow$$

$$Y(z) = -0.8z^{-1}Y(z) + X(z) \quad \Rightarrow \quad y(n) = -0.8y(n-1) + x(n)$$

The rest is as in problem (3c). Filter (e): With $a = -0.8$, the impulse response is finite:

$$h(n) = [1, a, a^2, a^3, a^4, a^5, a^6, a^7, 0, 0, 0, \cdots]$$

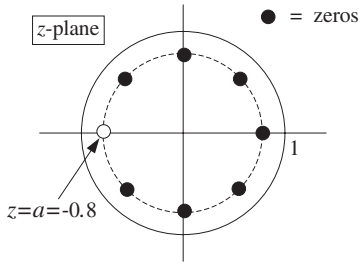
Its z-transform is

$$H(z) = 1 + az^{-1} + a^2z^{-2} + a^3z^{-3} + a^4z^{-4} + a^5z^{-5} + a^6z^{-6} + a^7z^{-7} = \frac{1 - a^8z^{-8}}{1 - az^{-1}}$$

The point $z = a$ is not really a pole of $H(z)$ because it is canceled by a numerator zero — $H(z)$ has seven zeros. To find them, solve the equation $z^8 = a^8$ and exclude the solution $z = a$, that is,

$$z = ae^{2\pi jk/8} = -0.8e^{2\pi jk/8} = 0.8e^{j\pi}e^{2\pi jk/8}, \quad k = 1, 2, 3, 4, 5, 6, 7$$

They are shown below

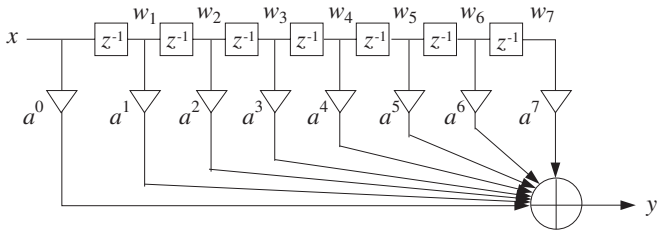


The I/O equation in the time domain is

$$y(n) = x(n) + ax(n-1) + a^2x(n-2) + \cdots + a^7x(n-7)$$

Sample-by-sample processing algorithm and block diagram realization:

for each input sample x do:
$y = x + aw_1 + a^2w_2 + a^3w_3 + a^4w_4 + a^5w_5 + a^6w_6 + a^7w_7$
$w_7 = w_6$
$w_6 = w_5$
$w_5 = w_4$
$w_4 = w_3$
$w_3 = w_2$
$w_2 = w_1$
$w_1 = x$

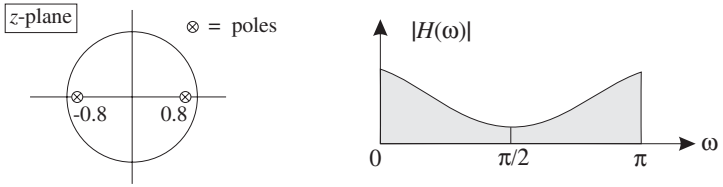


Filter (f): $H(z) = \dfrac{1}{1-0.8z^{-1}} + \dfrac{1}{1+0.8z^{-1}} = \dfrac{2}{1-0.64z^{-2}}$

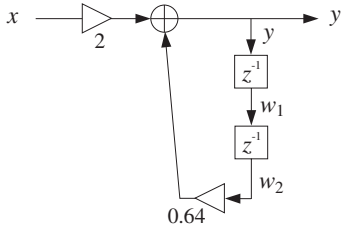
$$Y(z) = H(z)X(z) = \frac{2X(z)}{1-0.64z^{-2}} \quad \Rightarrow \quad Y(z) = 0.64z^{-2}Y(z) + 2X(z)$$

$$y(n) = 0.64y(n-2) + 2x(n)$$

There is a low frequency pole at $z = 0.8$ and a high frequency pole at $z = -0.8$. Thus, the filter will emphasize both the low and high frequencies:



The *direct form* realization and its sample processing algorithm are:



for each input sample x do:
$y = 2x + 0.64w_2$
$w_2 = w_1$
$w_1 = y$

Filters (g,h):

$$h(n) = 2(0.8)^n \cos(\pi n/2)u(n) = (0.8)^n 2\mathrm{Re}[j^n]u(n)$$

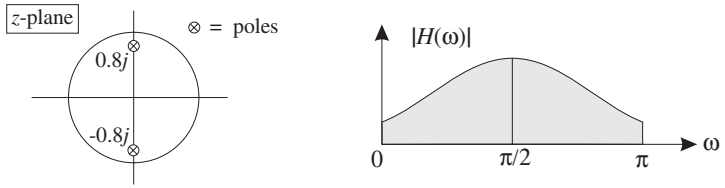
$$= (0.8j)^n u(n) + (-0.8j)^n u(n)$$

$$H(z) = \frac{1}{1-0.8jz^{-1}} + \frac{1}{1+0.8jz^{-1}} = \frac{2}{1+0.64z^{-2}}$$

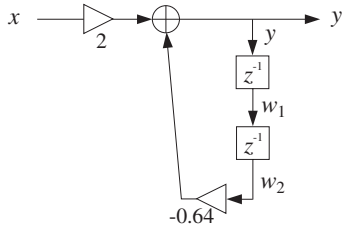
$$Y(z) = H(z)X(z) = \frac{2X(z)}{1+0.64z^{-2}} \quad \Rightarrow \quad Y(z) = -0.64z^{-2}Y(z) + 2X(z)$$

$$y(n) = -0.64y(n-2) + 2x(n)$$

There is a conjugate pair of poles at midfrequency, $z = \pm 0.8j = 0.8e^{\pm\pi/2}$. Thus, the filter will emphasize mid frequencies:



The *direct form* realization and its sample processing algorithm are:



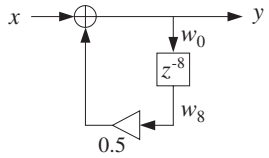
for each input sample x do:
$y = 2x - 0.64w_2$
$w_2 = w_1$
$w_1 = y$

Problem 6.6

Taking z-transforms, we find:

$$H(z) = 1 + 0.5z^{-8} + 0.5^2z^{-16} + 0.5^3z^{-24} + \cdots = \frac{1}{1-0.5z^{-8}}$$

where convergence requires $|z|^8 > 0.5$, or, $|z| > (0.5)^{1/8}$. The block diagram and sample processing algorithm are:



for each input sample x do:
$y = w_0 = x + 0.5w_8$
delay$(8, \mathbf{w})$

where the call to delay updates the 9-dimensional state vector of the eightfold delay $\mathbf{w} = [w_0, w_1, \ldots, w_8]$.

Problem 6.7

Because $z = e^{j\omega}$, we can deduce the transfer function from $H(\omega)$:

$$H(z) = \frac{-0.5 + z^{-8}}{1-0.5z^{-8}}$$

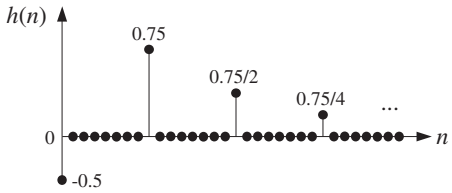
It can be written in a form that can be expanded in powers of z^{-8}:

$$H(z) = -0.5 + \frac{0.75z^{-8}}{1-0.5z^{-8}} = -0.5 + 0.75z^{-8}\left[1 + 0.5z^{-8} + 0.5^2z^{-16} + 0.5^3z^{-24} + \cdots\right]$$

Thus, the impulse response will be

$$\mathbf{h} = [-0.5, 0, 0, 0, 0, 0, 0, 0, 0.75, 0, 0, 0, 0, 0, 0, 0, 0.75(0.5), 0, 0, 0, 0, 0, 0, 0, 0.75(0.5)^2, \ldots]$$

It is depicted below:



Problem 6.8

a. The impulse response $h(n)$ satisfies the difference equation:

$$h(n) = h(n-1) + h(n-2) + \delta(n-1) \qquad \text{(P6.1)}$$

Indeed, for $n \geq 2$ the $\delta(n-1)$ term is absent and each $h(n)$ is the sum of the previous two. The sequence is properly initialized at $n = 0, 1$:

$$h(0) = h(-1) + h(-2) + \delta(-1) = 0 + 0 + 0 = 0$$
$$h(1) = h(0) + h(-1) + \delta(0) = 0 + 0 + 1 = 1$$

Next, we take z-transforms of both sides of Eq. (P6.1) to get $H(z)$:

$$H(z) = z^{-1}H(z) + z^{-2}H(z) + z^{-1}$$

because the z-transform of $\delta(n-1)$ is z^{-1}. Solving for $H(z)$:

$$H(z) = \frac{z^{-1}}{1 - z^{-1} - z^{-2}}$$

b. The poles of $H(z)$ are the roots of $1 - z^{-1} - z^{-2} = 0$ or

$$z^2 - z - 1 = 0 \qquad \Rightarrow \qquad z = \frac{1 \pm \sqrt{5}}{2}$$

Denoting the first one by ϕ, the other one will be:

$$\phi = \frac{1 + \sqrt{5}}{2}, \qquad -\frac{1}{\phi} = \frac{1 - \sqrt{5}}{2}$$

Indeed, we have

$$-\frac{1}{\phi} = -\frac{2}{1 + \sqrt{5}} = -\frac{2(1 - \sqrt{5})}{(1 + \sqrt{5})(1 - \sqrt{5})} = -\frac{2(1 - \sqrt{5})}{1 - 5} = \frac{1 - \sqrt{5}}{2}$$

Performing a partial fraction expansion on $H(z)$, we get:

$$H(z) = \frac{z^{-1}}{1 - z^{-1} - z^{-2}} = \frac{z^{-1}}{(1 - \phi z^{-1})(1 + \phi^{-1} z^{-1})} = \frac{A}{1 - \phi z^{-1}} + \frac{B}{1 + \phi^{-1} z^{-1}}$$

where

$$A = \left[\frac{z^{-1}}{1 + \phi^{-1} z^{-1}}\right]_{z=\phi} = \frac{\phi^{-1}}{1 + \phi^{-2}} = \frac{1}{\sqrt{5}}$$

$$B = \left[\frac{z^{-1}}{1 - \phi z^{-1}}\right]_{z=-\phi^{-1}} = -\frac{\phi}{1 + \phi^2} = -\frac{1}{\sqrt{5}}$$

Taking the causal inverse z-transform, we obtain a closed-form solution for the nth Fibonacci number:

$$h(n) = A\phi^n + B(-\phi)^{-n}, \qquad n \geq 0$$

It is unstable because ϕ is greater than 1 (note, $\phi \simeq 1.618$).

c. For large n, the exponentially increasing term ϕ^n dominates the expression. Therefore, the ratio of two successive Fibonacci numbers will tend to:

$$\frac{h(n+1)}{h(n)} \to \frac{A\phi^{n+1}}{A\phi^n} = \phi$$

This can also be seen from the difference equation of $h(n)$. For $n \geq 1$, we have

$$h(n+1) = h(n) + h(n-1)$$

Dividing by $h(n-1)$, we have:

$$\frac{h(n+1)}{h(n-1)} = \frac{h(n)}{h(n-1)} + 1 \qquad \text{or,}$$

$$\frac{h(n+1)}{h(n)} \cdot \frac{h(n)}{h(n-1)} = \frac{h(n)}{h(n-1)} + 1$$

If we denote the ratio $r(n) = h(n+1)/h(n)$, we may write this as:

$$r(n)r(n-1) = r(n-1) + 1$$

The limiting value $r(n) \to r$ will satisfy the limiting equation:

$$r^2 = r + 1$$

which is the same as that satisfied by ϕ. Thus, $r = \phi$ because they are the positive solutions.

d. The sequence $y(n)$ satisfies the recursion $y(n) = y(n-1) + y(n-2)$ for $n \geq 3$. Indeed, using the property $\phi^2 = \phi + 1$ and successively multiplying it by powers of ϕ, we get

$$y(3) = \phi^2 = \phi + 1 = y(2) + y(1)$$
$$y(4) = \phi^3 = \phi^2 + \phi = y(3) + y(2)$$
$$y(5) = \phi^4 = \phi^3 + \phi^2 = y(4) + y(3)$$
$$\ldots$$

The z-transform of $y(n)$ will be the once-delayed z-transform of the geometric series, that is,

$$Y(z) = \frac{z^{-1}}{1-\phi z^{-1}}$$

Thinking of $Y(z)$ as the output of $H(z)$ for a particular input $X(z)$, we have $Y(z) = H(z)X(z)$, which may be solved for $X(z)$:

$$X(z) = \frac{Y(z)}{H(z)} = \frac{\dfrac{z^{-1}}{1-\phi z^{-1}}}{\dfrac{z^{-1}}{(1-\phi z^{-1})(1+\phi^{-1}z^{-1})}} = 1+\phi^{-1}z^{-1}$$

which gives the sequence:

$$x(n) = \delta(n) + \phi^{-1}\delta(n-1) \qquad \text{or} \qquad \mathbf{x} = [1, \phi^{-1}, 0, 0, 0, 0 \dots]$$

Problem 6.9

As in the previous problem, the impulse response satisfies the difference equation:

$$h(n) = 2h(n-1) + h(n-2) + \delta(n-1)$$

with zero initial conditions. That is, the two start-up values are:

$$h(0) = 2h(-1) + h(-2) + \delta(-1) = 0 + 0 + 0 = 0$$
$$h(1) = 2h(0) + h(-1) + \delta(0) = 0 + 0 + 1 = 1$$

Taking z-transforms, we find $H(z)$:

$$H(z) = 2z^{-1}H(z) + z^{-2}H(z) + z^{-1} \qquad \Rightarrow \qquad H(z) = \frac{z^{-1}}{1-2z^{-1}-z^{-2}}$$

The roots of the denominator polynomial are the solutions of

$$z^2 = 2z + 1 \qquad \Rightarrow \qquad z = 1 \pm \sqrt{2} = \{\theta, -\theta^{-1}\}$$

The PF expansion is then

$$H(z) = \frac{z^{-1}}{1-2z^{-1}-z^{-2}} = \frac{z^{-1}}{(1-\theta z^{-1})(1+\theta^{-1}z^{-1})} = \frac{A}{1-\theta z^{-1}} + \frac{B}{1+\theta^{-1}z^{-1}}$$

where

$$A = -B = \frac{\theta}{\theta^2+1} = \frac{1}{2\sqrt{2}}$$

Thus, the nth Pell number will be:

$$h(n) = A\theta^n + B(-\theta)^{-n}, \qquad n \geq 0$$

Because the exponentially diverging term θ^n dominates the sum, the ratio of two successive numbers will converge to

$$\frac{h(n+1)}{h(n)} \to \frac{A\theta^{n+1}}{A\theta^n} = \theta$$

The $y(n)$ sequence has z-transform

$$Y(z) = \frac{z^{-1}}{1-\theta z^{-1}} = \frac{z^{-1}}{(1-\theta z^{-1})(1+\theta^{-1}z^{-1})} \cdot (1+\theta^{-1}z^{-1}) = H(z)(1+\theta^{-1}z^{-1})$$

and therefore, it may be thought of as the output when the input is:

$$X(z) = 1 + \theta^{-1}z^{-1}$$

or,

$$x(n) = \delta(n) + \theta^{-1}\delta(n-1) \qquad \Rightarrow \qquad \mathbf{x} = [1, \theta^{-1}, 0, 0, 0 \dots]$$

Problem 6.10

The input and output z-transforms are:

$$X_1(z) = \frac{1}{1-0.5z^{-1}}, \qquad Y_1(z) = \frac{1}{1-0.5z^{-1}} + \frac{1}{1-0.4z^{-1}} = \frac{2(1-0.45z^{-1})}{(1-0.5z^{-1})(1-0.4z^{-1})}$$

It follows that transfer function is:

$$H(z) = \frac{Y_1(z)}{X_1(z)} = \frac{2(1-0.45z^{-1})}{1-0.4z^{-1}}$$

Therefore, the second output, having z-transform

$$Y_2(z) = \frac{1}{1-0.4z^{-1}}$$

will be produced by

$$X_2(z) = \frac{Y_2(z)}{H(z)} = \frac{0.5}{1-0.45z^{-1}} \qquad \Rightarrow \qquad x_2(n) = 0.5(0.45)^n u(n)$$

Problem 6.11

The input and output z-transforms are:

$$X_1(z) = \frac{1}{1-az^{-1}}$$

$$Y_1(z) = \frac{1}{1-az^{-1}} + \frac{1}{1-bz^{-1}} = \frac{2-(a+b)z^{-1})}{(1-az^{-1})(1-bz^{-1})} = \frac{2(1-cz^{-1})}{(1-az^{-1})(1-bz^{-1})}$$

where $c = (a+b)/2$. Therefore, the transfer function is

$$H(z) = \frac{Y_1(z)}{X_1(z)} = \frac{2(1-cz^{-1})}{1-bz^{-1}}$$

If the input $x_2(n) = c^n u(n)$ is applied it will have z-transform $X_2(z) = 1/(1-cz^{-1})$ and will cause the output:

$$Y_2(z) = H(z)X_2(z) = \frac{2(1-cz^{-1})}{1-bz^{-1}} \cdot \frac{1}{1-cz^{-1}} = \frac{2}{1-bz^{-1}}$$

and in the time domain:

$$y_2(n) = 2b^n u(n)$$

Problem 6.12

The input and output z-transforms are:

$$X(z) = \frac{1}{1-0.7z^{-1}}$$

$$Y(z) = \frac{1}{1-0.7z^{-1}} + \frac{1}{1-0.5z^{-1}} = \frac{2(1-0.6z^{-1})}{(1-0.7z^{-1})(1-0.5z^{-1})}$$

Therefore the transfer function will be:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{2(1-0.6z^{-1})}{1-0.5z^{-1}}$$

Expanding in partial fractions, we have:

$$H(z) = \frac{2(1-0.6z^{-1})}{1-0.5z^{-1}} = 2.4 - \frac{0.4}{1-0.5z^{-1}}$$

which has the causal inverse:

$$h(n) = 2.4\delta(n) - 0.4(0.5)^n u(n)$$

Problem 6.13

The normalized peak and width frequencies are:

$$\omega_0 = \frac{2\pi f_0}{f_s} = \frac{2\pi 250}{5000} = 0.1\pi \text{ [rads/sample]}$$

$$\Delta\omega = \frac{2\pi\Delta f}{f_s} = \frac{2\pi 20}{5000} = 0.008\pi \text{ [rads/sample]}$$

Therefore, the Q-factor of the filter will be:

$$Q = \frac{\omega_0}{\Delta\omega} = \frac{0.1\pi}{0.008\pi} = 12.5$$

We take the filter poles to be at locations:

$$p = Re^{j\omega_0}, \qquad p^* = Re^{-j\omega_0}$$

The pole radius R may be determined from the approximate relationship

$$\Delta\omega = 2(1-R)$$

which gives:

$$R = 1 - \frac{1}{2}\Delta\omega = 1 - \frac{1}{2}0.008\pi = 0.9874$$

The denominator polynomial will be:

$$1 + a_1 z^{-1} + a_2 z^{-2} = (1-pz^{-1})(1-p^*z^{-1}) = 1 - (p+p^*)z^{-1} + p^*pz^{-2}$$

which gives:

$$a_1 = -(p+p^*) = -2R\cos\omega_0 = -2\cdot 0.9874\cdot\cos(0.1\pi) = -1.8781$$

$$a_2 = p^*p = R^2 = 0.9750$$

Therefore, the designed filter will be:

$$H(z) = \frac{1}{1-1.8781z^{-1}+0.9750z^{-2}}$$

The time constant of the resonator is given in terms of the maximum pole radius (there is only one radius here):

$$n_{\text{eff}} = \frac{\ln\epsilon}{\ln R} = \frac{\ln(0.01)}{\ln(0.9874)} = 363.18$$

The approximate expression is obtained by replacing R in terms of $\Delta\omega$:

$$n_{\text{eff}} = \frac{\ln\epsilon}{\ln R} = \frac{\ln\epsilon}{\ln(1-\Delta\omega/2)} \simeq \frac{\ln\epsilon}{-\Delta\omega/2} = -\frac{2\ln\epsilon}{\Delta\omega}$$

where we used the small-x approximation $\ln(1-x)\simeq -x$. The approximate numerical value of the time constant will be:

$$n_{\text{eff}} = -\frac{2\ln\epsilon}{\Delta\omega} = -\frac{2\ln(0.01)}{0.008\pi} = 366.47$$

which compares well with the exact value.

Problem 6.14

The ϵ-level time constant of a filter is (in sampling instants):

$$n_{\text{eff}} = \frac{\ln\epsilon}{\ln\rho}$$

where $\rho = \max_i |p_i|$ is the maximum pole radius. In seconds, the time constant is:

$$\tau = n_{\text{eff}}T = \frac{\ln\epsilon}{\ln\rho}T$$

where $T = 1/f_s$ is the interval between samples. It follows that the ϵ_1 and ϵ_2 level time constants will be

$$\tau_1 = \frac{\ln\epsilon_1}{\ln\rho}T, \qquad \tau_2 = \frac{\ln\epsilon_2}{\ln\rho}T$$

And therefore,

$$\tau_1 = \frac{\ln\epsilon_1}{\ln\epsilon_2}\tau_2$$

In particular, for the 60 versus 40 dB time constants, the scale factor is:

$$\frac{\ln\epsilon_1}{\ln\epsilon_2} = \frac{\ln(0.001)}{\ln(0.01)} = \frac{3}{2} = 1.5$$

Problem 6.15

Using the approximate expression of Problem 6.13 and $\epsilon = 0.001$, we have:

$$\tau = n_{\text{eff}}T = -\frac{2\ln\epsilon}{\Delta\omega}T = -\frac{2\ln\epsilon}{2\pi\Delta f T}T = -\frac{\ln\epsilon}{\pi\Delta f}$$

which gives:

$$\tau = -\frac{\ln\epsilon}{\pi\Delta f} - \frac{\ln(0.001)}{\pi\Delta f} = \frac{2.2}{\Delta f}$$

Problem 6.16

The periodic impulse response may be thought of as the sum of the delayed replicas of one period, that is,

$$h(n) = g(n) + g(n-8) + g(n-16) + \cdots$$

where $g(n) = [1, 2, 3, 4, 0, 0, 0, 0]$ is the basic period of length 8. Taking z-transforms we obtain:

$$H(z) = G(z) + z^{-8}G(z) + z^{-16}G(z) + \cdots = (1 + z^{-8} + z^{-16} + \cdots)G(z)$$

where $G(z) = 1 + 2z^{-1} + 3z^{-2} + 4z^{-3}$. Using the geometric series on the first factor we obtain:

$$H(z) = \frac{G(z)}{1 - z^{-8}} = \frac{1 + 2z^{-1} + 3z^{-2} + 4z^{-3}}{1 - z^{-8}}$$

The direct and canonical realizations are shown in Fig. P6.1. Note that in the canonical realization the total number of delays is 8, but the last 5 are lumped together into a fivefold delay z^{-5}. The sample processing algorithm for the canonical case is:

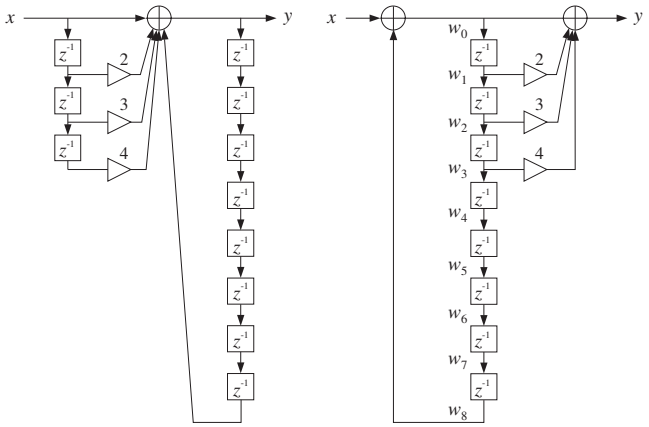


Fig. P6.1 Filter realizations of Problem 6.16.

```
for each input x do:
    w0 = x + w8
    y = w0 + 2w1 + 3w2 + 4w3
    delay(8, w)
```

where the call to delay$(8, \mathbf{w})$ implements the updating of the delay line, that is, $w_8 = w_7$, $w_7 = w_6$, $\ldots$, $w_1 = w_0$.

Problem 6.17

The solution is similar to that of Problem 6.16. Here, the repetition period is 4 and we have:

$$h(n) = g(n) + g(n-4) + g(n-8) + g(n-12) + \cdots$$

where $g(n) = [0, 1, 2, 3]$. In the z-domain, we have:

$$H(z) = G(z) + z^{-4}G(z) + z^{-8}G(z) + \cdots = (1 + z^{-4} + z^{-8} + \cdots)G(z) = \frac{G(z)}{1 - z^{-4}}$$

where $G(z) = z^{-1} + 2z^{-2} + 3z^{-3}$. Thus,

$$H(z) = \frac{z^{-1} + 2z^{-2} + 3z^{-3}}{1 - z^{-4}}$$

The direct and canonical realizations are shown in Fig. P6.2.

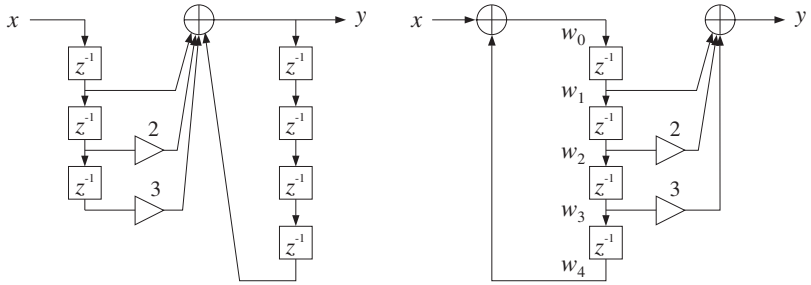


Fig. P6.2 Direct and canonical realizations of Problem 6.17.

The sample processing algorithm for the canonical case is:

```
for each input x do:
    w0 = x + w4
    y = w1 + 2w2 + 3w3
    delay(4, w)
```

where the call to delay$(4, \mathbf{w})$ implements the updating of the delay line, that is, $w_4 = w_3$, $w_3 = w_2$, $\ldots$, $w_1 = w_0$. The cascade realization is obtained by factoring the numerator and denominator into (up to) 2nd order polynomials:

$$H(z) = \frac{z^{-1} + 2z^{-2} + 3z^{-3}}{1 - z^{-4}} = \left[\frac{z^{-1}}{1 - z^{-2}}\right]\left[\frac{1 + 2z^{-1} + 3z^{-2}}{1 + z^{-2}}\right] \equiv H_0(z)H_1(z)$$

The block diagram is shown in Fig. P6.3. The corresponding sample processing algorithm is:

```
for each input x do:
    w0 = x + w2
    x1 = w1
    w2 = w1
    w1 = w0
    v0 = x1 − v2
    y = v0 + 2v1 + 3v3
    v2 = v1
    v1 = v0
```

Using partial fractions, we may write:

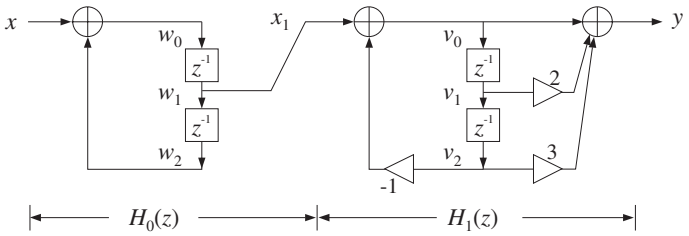


Fig. P6.3 Cascade realization of Problem 6.17.

$$H(z) = \frac{z^{-1}+2z^{-2}+3z^{-3}}{1-z^{-4}} = \frac{z^{-1}+2z^{-2}+3z^{-3}}{(1-z^{-1})(1+z^{-1})(1-jz^{-1})(1+jz^{-1})}$$
$$= \frac{A}{1-z^{-1}} + \frac{B}{1+z^{-1}} + \frac{C-jD}{1-jz^{-1}} + \frac{C+jD}{1+jz^{-1}}$$

where the PFE coefficients $C \pm jD$ are conjugates. The numerical values are:

$$A = \left[\frac{z^{-1}+2z^{-2}+3z^{-3}}{(1+z^{-1})(1-jz^{-1})(1+jz^{-1})}\right]_{z=1} = 1.5$$

$$B = \left[\frac{z^{-1}+2z^{-2}+3z^{-3}}{(1-z^{-1})(1-jz^{-1})(1+jz^{-1})}\right]_{z=-1} = -0.5$$

$$C-jD = \left[\frac{z^{-1}+2z^{-2}+3z^{-3}}{(1-z^{-1})(1+z^{-1})(1+jz^{-1})}\right]_{z=j} = -0.5+j0.5$$

Taking (causal) inverse z-transforms, we find for $n \geq 0$:

$$h(n) = A + B(-1)^n + (C-jD)j^n + (C+jD)(-j)^n = A + B(-1)^n + 2\mathrm{Re}\left[(C-jD)e^{j\pi n/2}\right]$$
$$= A + B(-1)^n + 2C\cos(\pi n/2) + 2D\sin(\pi n/2)$$
$$= 1.5 - 0.5(-1)^n - \cos(\pi n/2) - \sin(\pi n/2)$$

Evaluating at the first couple of periods, we find:

$$h(n) = [0,1,2,3,0,1,2,3,\dots]$$

Problem 6.18

Expand the denominator in powers of z^{-7} and pick out the coefficients of the powers of z^{-1}. We have:

$$H(z) = \frac{1+z^{-1}+z^{-2}+z^{-3}}{1-z^{-7}} = (1+z^{-1}+z^{-2}+z^{-3})(1+z^{-7}+z^{-14}+\cdots)$$

The powers of z^{-7} cause the period-7 replication of the basic period $G(z) = 1+z^{-1}+z^{-2}+z^{-3}$:

$$h(n) = [\underbrace{1,1,1,1,0,0,0}, \underbrace{1,1,1,1,0,0,0}, \underbrace{1,1,1,1,0,0,0}, \dots]$$

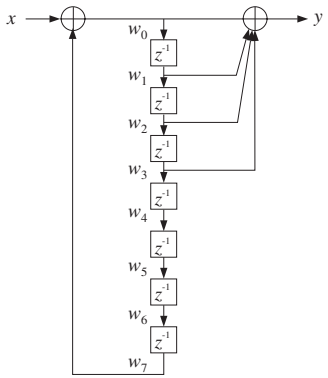


Fig. P6.4 Canonical realizations of Problem 6.18.

The canonical realization is shown in Fig. P6.4.
The sample processing algorithm is:

```
for each input x do:
    w0 = x + w7
    y = w0 + w1 + w2 + w3
    delay(7, w)
```

where the call to delay implements the delay-line updates $w_7 = w_6$, $w_6 = w_5$, …, $w_1 = w_0$. The output due to the input $\mathbf{x} = [3,2,1]$ may be obtained by using the LTI form of convolution, that is, summing up the delayed replicas of the impulse response, scaled by the input samples:

$$x_0 h_n = [3,3,3,3,0,0,0,3,3,3,3,0,0,0,3,3,3,3,0,0,0,\dots]$$
$$x_0 h_{n-1} = [0,2,2,2,2,0,0,0,2,2,2,2,0,0,0,2,2,2,2,0,0,\dots]$$
$$x_0 h_{n-2} = [0,0,1,1,1,1,0,0,0,1,1,1,1,0,0,0,1,1,1,1,0,\dots]$$
$$y_n = [3,5,6,6,3,1,0,3,5,6,6,3,1,0,3,5,6,6,3,1,0,\dots]$$

Thus, the output also has period 7. The same conclusion can be reached working with z-transforms. We have for the output z-transform:

$$Y(z) = H(z)X(z) = \frac{(1+z^{-1}+z^{-2}+z^{-3})(3+2z^{-1}+z^{-2})}{1-z^{-7}}$$
$$= \frac{3+5z^{-1}+6z^{-2}+6z^{-3}+3z^{-4}+z^{-5}}{1-z^{-7}}$$

The expansion of the denominator in powers of z^{-7} will cause the period-7 replication of the numerator.

Problem 6.19

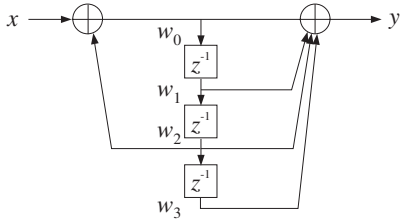
The expansion of the denominator, will cause the period-2 replication of the numerator. Because the numerator has length 4, its period-2 replicas will overlap with each other and must be added together:

$$H(z) = \frac{1 + z^{-1} + z^{-2} + z^{-3}}{1 - z^{-2}} = (1 + z^{-1} + z^{-2} + z^{-3})(1 + z^{-2} + z^{-4} + \cdots)$$

or, in the time domain:

$$\begin{aligned}\mathbf{h} &= [1,1,1,1,0,0,0,0,0,0,0,0,\ldots] \\ &+ [0,0,1,1,1,1,0,0,0,0,0,0,\ldots] \\ &+ [0,0,0,0,1,1,1,1,0,0,0,0,\ldots] + \cdots \\ &= [1,1,2,2,2,2,2,2,2,2,2,2,\ldots]\end{aligned}$$

The canonical realization and sample processing algorithm are:



for each input x do:
$w_0 = x + w_2$
$y = w_0 + w_1 + w_2 + w_3$
$w_3 = w_2$
$w_2 = w_1$
$w_1 = w_0$

Problem 6.20

Writing the sample processing algorithm in terms of the time index n, we have

$$\begin{aligned} w_0(n) &= x(n) + w_1(n) \\ y(n) &= w_0(n) + w_2(n) \\ w_2(n+1) &= w_1(n) \\ w_1(n+1) &= w_0(n) \end{aligned}$$

or, in the z-domain

$$\begin{aligned} W_0(z) &= X(z) + W_1(z) \\ Y(z) &= W_0(z) + W_2(z) \\ zW_2(z) &= W_1(z) \\ zW_1(z) &= W_0(z) \end{aligned}$$

Eliminating $W_0(z), W_1(z), W_2(z)$ in favor of $Y(z)$ and $X(z)$, we obtain:

$$W_1(z) = z^{-1}W_0(z), \qquad W_2(z) = z^{-1}W_1(z) = z^{-2}W_0(z)$$

$$W_0(z) = X(z) + z^{-1}W_0(z) \qquad \Rightarrow \qquad W_0(z) = \frac{X(z)}{1 - z^{-1}}$$

$$Y(z) = W_0(z) + z^{-2}W_0(z) = (1 + z^{-2})W_0(z) = (1 + z^{-2})\frac{X(z)}{1 - z^{-1}}$$

which gives:

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

Multiplying numerator and denominator by $1 + z^{-1}$, we get:

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

which is the same as that of Problem 6.19.

Problem 6.21

The number of harmonics fitting within the Nyquist interval are:

$$\frac{f_s}{f_1} = \frac{240}{60} = 4$$

Therefore, the harmonics will lie at the fourth roots of unity around the unit circle:

$$\omega_k = k\omega_1 = \frac{2\pi k}{4}, \qquad k = 0,1,2,3$$

The numerator polynomial $1 - z^{-4}$ will have $e^{j\omega_k}$ as roots. Putting the poles just behind the zeros, we get the desired multinotch filter:

$$H(z) = \frac{1 - z^{-4}}{1 - az^{-4}}$$

where $0 < a < 1$ must be chosen to be very near 1.

Problem 6.22

The zeros are at the 16th roots of unity, that is,

$$z_k = e^{j\omega_k} = e^{2\pi jk/16}, \qquad k = 0,1,\ldots,15$$

The poles are just behind the zeros at the same angles:

$$p_k = a^{1/16}e^{j\omega_k} = a^{1/16}e^{2\pi jk/16}, \qquad k = 0,1,\ldots,15$$

The zeros and poles are shown in Fig. P6.5. If a is very near 1, the magnitude response will vanish at the harmonics $\omega_k = k\omega_1 = 2\pi k/16$, and be essentially flat between zeros. Fig. P6.6 shows a sketch of $|H(\omega)|$.
The impulse response is obtained by expanding $H(z)$ into powers of z^{-16}. We have with $A = 1/a$ and $B = 1 - A = 1 - 1/a$:

$$H(z) = \frac{1 - z^{-16}}{1 - az^{-16}} = A + \frac{B}{1 - az^{-16}} = A + B(1 + az^{-16} + a^2z^{-32} + \cdots)$$

Thus, the causal impulse response will be $h(n)$:

$$h(n) = \begin{cases} A + B = 1, & \text{if } n = 0 \\ Ba^{n/16}, & \text{if } n \text{ is a non-zero multiple of 16} \\ 0, & \text{otherwise} \end{cases}$$

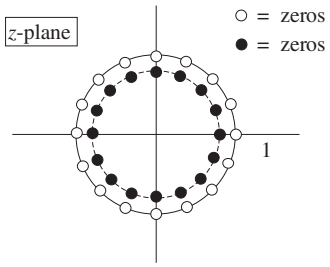


Fig. P6.5 Pole/zero pattern of Problem 6.22.

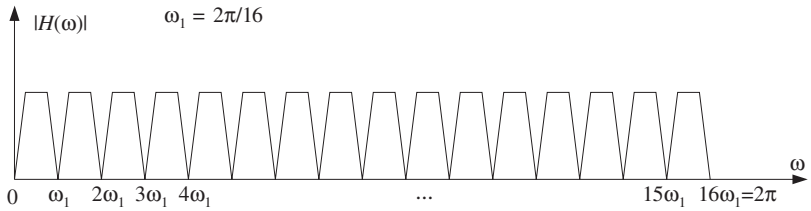
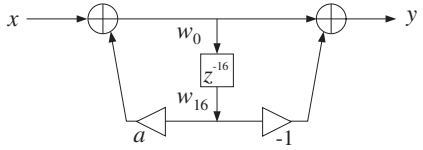


Fig. P6.6 Magnitude response of Problem 6.22.

The canonical realization and sample processing algorithm are:



for each input x do:
$w_0 = x + w_{16}$
$y = w_0 - aw_{16}$
delay(16, **w**)

where the call to delay updates the 17-dimensional delay line.

Problem 6.23

$H(z) = -0.3 + \dfrac{0.6}{1 - 0.5z^{-1}}$. It follows that $h(n) = -0.3\delta(n) + 0.6(0.5)^n u(n)$.

Problem 6.24

The gain factor $1 - a$ normalizes the magnitude response to unity at DC. The direct form I/O difference equation is obtained from:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1-a}{1-az^{-1}} \qquad \Rightarrow \qquad (1-az^{-1})Y(z) = (1-a)X(z)$$

or,

$$Y(z) = az^{-1}Y(z) + (1-a)X(z) = X(z) + a(z^{-1}Y(z) - X(z))$$

and in the time domain:

$$y(n) = x(n) + a(y(n-1) - x(n))$$

A realization based on this expression, as well as the canonical realization, are shown in Fig. 6.24. The corresponding sample processing algorithm will be:

for each input x do:
$y = x + a(w_1 - x)$
$w_1 = y$

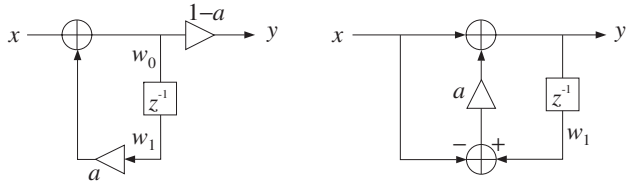


Fig. P6.7 Realizations of Problem 6.24.

Problem 6.25

The filter poles are at $z = 2, -0.5$. Thus,

$$H(z) = \frac{3 - 3z^{-1} - z^{-2}}{1 - 1.5z^{-1} - z^{-2}} = \frac{3 - 3z^{-1} - z^{-2}}{(1 - 2z^{-1})(1 + 0.5z^{-1})}$$

Expanding in partial fractions, we have:

$$H(z) = \frac{3 - 3z^{-1} - z^{-2}}{(1 - 2z^{-1})(1 + 0.5z^{-1})} = A + \frac{B}{1 - 2z^{-1}} + \frac{C}{1 + 0.5z^{-1}}$$

where

$$A = \left[\frac{3 - 3z^{-1} - z^{-2}}{1 - 1.5z^{-1} - z^{-2}}\right]_{z=0} = 1$$

$$B = \left[\frac{3 - 3z^{-1} - z^{-2}}{1 + 0.5z^{-1}}\right]_{z=2} = 1$$

$$C = \left[\frac{3 - 3z^{-1} - z^{-2}}{1 - 2z^{-1}}\right]_{z=-0.5} = 1$$

The three possible ROCs and corresponding inverses are:

$$\begin{aligned} |z| > 2 \quad , \quad & h(n) = A\delta(n) + B2^n u(n) + C(-0.5)^n u(n) \\ 2 > |z| > 0.5, \quad & h(n) = A\delta(n) - B2^n u(-n-1) + C(-0.5)^n u(n) \\ |z| < 0.5, \quad & h(n) = A\delta(n) - B2^n u(-n-1) - C(-0.5)^n u(-n-1) \end{aligned}$$

Only the second one is stable. Its ROC contains the unit circle.

Problem 6.26

The transfer function is obtained from the I/O difference equation by taking z-transforms of both sides and solving for the ratio $Y(z)/X(z)$:

$$Y(z) = 2.5z^{-1}Y(z) - z^{-2}Y(z) + 3X(z) + 3z^{-2}X(z) \quad \Rightarrow \quad H(z) = \frac{Y(z)}{X(z)} = \frac{3(1+z^{-2})}{1-2.5z^{-1}+z^{-2}}$$

Factoring the denominator into its poles and expanding in partial fractions, we get:

$$H(z) = \frac{3(1+z^{-2})}{1-2.5z^{-1}+z^{-2}} = \frac{3(1+z^{-2})}{(1-2z^{-1})(1-0.5z^{-1})} = A + \frac{B}{1-2z^{-1}}\frac{C}{1-0.5z^{-1}}$$

where

$$A = \left[\frac{3(1+z^{-2})}{1-2.5z^{-1}+z^{-2}}\right]_{z=0} = 3$$

$$B = \left[\frac{3(1+z^{-2})}{1-0.5z^{-1}}\right]_{z=2} = 5$$

$$C = \left[\frac{3(1+z^{-2})}{1-2z^{-1}}\right]_{z=0.5} = -5$$

Thus, the ROCs and corresponding inverses are:

$$\begin{aligned} |z| > 2 \quad , \quad & h(n) = A\delta(n) + B2^n u(n) + C0.5^n u(n) \\ 2 > |z| > 0.5, \quad & h(n) = A\delta(n) - B2^n u(-n-1) + C0.5^n u(n) \\ |z| < 0.5, \quad & h(n) = A\delta(n) - B2^n u(-n-1) - C0.5^n u(-n-1) \end{aligned}$$

The z-transform of the signal $g(n)$ is:

$$g(n) = \cos(\pi n/2)u(n) \xrightarrow{z} G(z) = \frac{1}{1+z^{-2}}$$

Therefore, the z-transform of the input $x(n)$ will be:

$$x(n) = g(n) - 2g(n-1) \xrightarrow{z} X(z) = G(z) - 2z^{-1}G(z) = (1-2z^{-1})G(z) = \frac{1-2z^{-1}}{1+z^{-2}}$$

Thus, the output z-transform will be:

$$Y(z) = H(z)X(z) = \frac{3(1+z^{-2})}{(1-2z^{-1})(1-0.5z^{-1})} \cdot \frac{1-2z^{-1}}{1+z^{-2}} = \frac{3}{1-0.5z^{-1}}$$

Inverting causally, we find:

$$y(n) = 3(0.5)^n u(n)$$

Problem 6.27

Using the inverse DTFT formula of Eq. (6.3.5), we have:

$$y(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H(\omega)X(\omega)e^{j\omega n}\,d\omega$$

Similarly, for the input signal, assuming its spectrum is restricted to be only over the range $[-\omega_c, \omega_c]$:

$$x(n) = \frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} X(\omega)e^{j\omega n}\,d\omega$$

Using $d(\omega) = D$, the definition of $H(\omega)$ over one Nyquist interval will be:

$$H(\omega) = \begin{cases} Ge^{-jD\omega}, & \text{for } 0 \le |\omega| \le \omega_c \\ 0, & \text{for } \omega_c < |\omega| \le \pi \end{cases}$$

where G is its passband gain. Thus, the integration range for $y(n)$ also collapses to $[-\omega_c, \omega_c]$, giving:

$$y(n) = \frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} Ge^{-jD\omega}X(\omega)e^{j\omega n}\,d\omega = G\frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} X(\omega)e^{j\omega(n-D)}\,d\omega$$

Comparing this with the expression of $x(n)$, we recognize that

$$y(n) = Gx(n-D)$$

Problem 6.28

The PF expansion of $H(z)$ is of the form:

$$H(z) = \frac{N(z)}{\prod_{i=1}^{M}(1-p_i z^{-1})} = \sum_{i=1}^{M}\frac{A_i}{1-p_i z^{-1}}$$

where the i-th coefficient is obtained by deleting the factor $(1-p_i z^{-1})$ from the left-hand side and evaluating the remainder at $z = p_i$, that is,

$$A_i = \left[\frac{N(z)}{\prod_{j\neq i}(1-p_j z^{-1})}\right]_{z=p_i} = \frac{N(p_i)}{\prod_{j\neq i}(1-p_j p_i^{-1})}$$

Taking causal inverse z-transforms, we find for $h(n)$:

$$h(n) = \sum_{i=1}^{M} A_i p_i^n u(n)$$

Problem 6.29

The z-transforms of the causal and anticausal sinusoidal inputs are the same up to a negative sign:

$$e^{j\omega_0 n}u(n) \xrightarrow{z} \frac{1}{1-e^{j\omega_0}z^{-1}}$$

$$e^{j\omega_0 n}u(-n-1) \xrightarrow{z} -\frac{1}{1-e^{j\omega_0}z^{-1}}$$

with ROCs $|z| > 1$ and $|z| < 1$, respectively. The output z-transform will be $Y(z) = H(z)X(z)$. Thus, in the two cases:

$$Y(z) = \pm \frac{N(z)}{(1 - e^{j\omega_0} z^{-1}) \prod_{i=1}^{M} (1 - p_i z^{-1})}$$

The partial fraction expansion leads to

$$Y(z) = \pm \frac{C}{1 - e^{j\omega_0} z^{-1}} \pm \sum_{i=1}^{M} \frac{B_i}{1 - p_i z^{-1}}$$

where $C = H(\omega_0)$, as we saw in Eq. (6.3.10), and B_i are obtained by deleting the factor $(1 - p_i z^{-1})$ and evaluating the rest at $z = p_i$:

$$B_i = \left[\frac{N(z)}{(1 - e^{j\omega_0} z^{-1}) \prod_{j \neq i} (1 - p_j z^{-1})} \right]_{z = p_i} = \frac{N(p_i)}{(1 - e^{j\omega_0} p_i^{-1}) \prod_{j \neq i}^{M} (1 - p_j p_i^{-1})}$$

Comparing with the result of the previous problem, we have:

$$B_i = \frac{A_i}{1 - e^{j\omega_0} p_i^{-1}}$$

To get the inverse z-transform $y(n)$, we must assume a particular ROC. For the causal case, we take the ROC to be $|z| > 1$, so that all the PFE terms will be inverted causally. For the anticausal case, we take the filter part to be causal and the input anticausal, that is, the ROC will be the annular region between the maximum pole and the unit circle:

$$\max_i |p_i| < |z| < 1$$

In this case the C term will be inverted anticausally and the B_i terms causally. These choices for the ROCs, give:

$$y(n) = H(\omega_0) e^{j\omega_0 n} u(n) + \sum_{i=1}^{M} B_i p_i^n u(n)$$

$$y(n) = H(\omega_0) e^{j\omega_0 n} u(-n-1) - \sum_{i=1}^{M} B_i p_i^n u(n)$$

Adding up the right-sided and left-sided sinusoidal inputs gives rise to the double-sided input:

$$e^{j\omega_0 n} u(n) + e^{j\omega_0 n} u(-n-1) = e^{j\omega_0 n} \big(u(n) + u(-n-1) \big) = e^{j\omega_0 n}, \qquad -\infty < n < \infty$$

where, we used the property $u(n) + u(-n-1) = 1$ for all n. It follows from the linearity of the filter that the corresponding output will be the sum of the two outputs obtained above. This gives:

$$\begin{aligned} y(n) &= H(\omega_0) e^{j\omega_0 n} u(n) + \sum_{i=1}^{M} B_i p_i^n u(n) + H(\omega_0) e^{j\omega_0 n} u(-n-1) - \sum_{i=1}^{M} B_i p_i^n u(n) \\ &= H(\omega_0) e^{j\omega_0 n} u(n) + H(\omega_0) e^{j\omega_0 n} u(-n-1) \\ &= H(\omega_0) e^{j\omega_0 n} \end{aligned}$$

which is recognized as the standard steady-state sinusoidal response of the filter.

Problem 6.30

The PF expansion of $H(z)$ gives:

$$H(z) = \frac{3 - 5z^{-1} + z^{-2}}{(1 - 0.5z^{-1})(1 - 2z^{-1})} = 1 + \frac{1}{1 - 0.5z^{-1}} + \frac{1}{1 - 2z^{-1}}$$

The stable ROC containing the unit circle will be $0.5 < |z| < 2$. Therefore, the 0.5-pole term will be inverted causally and the 2-pole anticausally:

$$h(n) = \delta(n) + 0.5^n u(n) - 2^n u(-n-1)$$

The truncated impulse response agrees with $h(n)$ for $n \geq -D$ and is zero otherwise, that is:

$$\widetilde{h}(n) = \begin{cases} \delta(n) + 0.5^n, & n \geq 0 \\ -2^n, & -D \leq n \leq -1 \\ 0, & n \leq -D-1 \end{cases}$$

The corresponding z-transform is by definition:

$$\widetilde{H}(z) = \sum_{n=0}^{\infty} [\delta(n) + 0.5^n] z^{-n} - \sum_{n=-D}^{-1} 2^n z^{-n}$$

The last term can be summed by changing summation variables from n to $m = -n$:

$$\sum_{n=-D}^{-1} 2^n z^{-n} = \sum_{m=1}^{D} 2^{-m} z^m = \frac{2^{-1} z (1 - 2^{-D} z^D)}{1 - 2^{-1} z} = -\frac{1 - 2^{-D} z^D}{1 - 2z^{-1}}$$

where we used the finite geometric series:

$$\sum_{m=1}^{D} x^m = x + x^2 + \cdots x^D = x(1 + x + x^2 + \cdots x^{D-1}) = \frac{x(1 - x^D)}{1 - x}$$

Thus,

$$\widetilde{H}(z) = 1 + \frac{1}{1 - 0.5z^{-1}} + \frac{1 - 2^{-D} z^D}{1 - 2z^{-1}}$$

Comparing with the exact $H(z)$ we have:

$$\widetilde{H}(z) = H(z) - \frac{2^{-D} z^D}{1 - 2z^{-1}}$$

or, the error transfer function:

$$H(z) - \widetilde{H}(z) = \frac{2^{-D} z^D}{1 - 2z^{-1}}$$

The given input has z-transform:

$$x(n) = \delta(n) - 2\delta(n-1) \xrightarrow{Z} X(z) = 1 - 2z^{-1}$$

The error output will be

$$E(z) = Y(z) - \widetilde{Y}(z) = H(z) X(z) - \widetilde{H}(z) X(z) = \big(H(z) - \widetilde{H}(z) \big) X(z)$$

or,

$$E(z) = \frac{2^{-D} z^D}{1 - 2z^{-1}} \cdot (1 - 2z^{-1}) = 2^{-D} z^D$$

and in the time domain:

$$e(n) = 2^{-D} \delta(n + D)$$

By choosing D large enough, the factor 2^{-D} can be made as small as desired.

Chapter 7 Problems

Problem 7.1

Expanding the denominator in powers of z^{-5} will replicate the numerator periodically with period 5. But because the numerator itself has length 5, the replicas will not overlap, and we will get the repetition of the numerator coefficients:

$$\mathbf{h} = [\underbrace{0,1,2,3,4},\underbrace{0,1,2,3,4},\underbrace{0,1,2,3,4},\ldots]$$

The direct and canonical realizations are shown in Fig. P7.1.

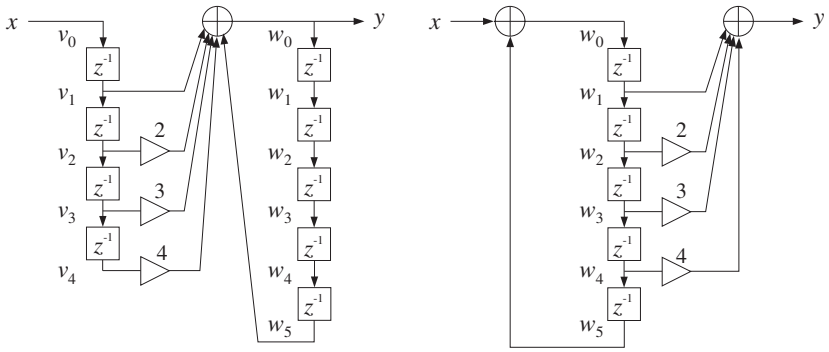


Fig. P7.1 Direct and canonical realizations of Problem 7.1.

The sample processing algorithms are expressed with the help of the routine *delay*, which updates the filter's internal state vectors.

for each input x do:
$v_0 = x$
$w_0 = w_5 + v_1 + 2v_2 + 3v_3 + 4v_4$
$y = w_0$
delay$(4, \mathbf{v})$
delay$(5, \mathbf{w})$

for each input x do:
$w_0 = x + w_5$
$y = w_1 + 2w_2 + 3w_3 + 4w_4$
delay$(5, \mathbf{w})$

where $\mathbf{v} = [v_0, v_1, v_2, v_4]$, $\mathbf{w} = [w_0, w_1, w_2, w_3, w_4.w_5]$ are the 5- and 6-dimensional state vectors needed for the numerator and denominator polynomials. The calls to the *delay* routines update the delay line, e.g., the call *delay*$(5, \mathbf{w})$ is equivalent to the shifts: $w_5 = w_4$, $w_4 = w_3$, $w_3 = w_2$, $w_2 = w_1$, $w_1 = w_0$.

Factoring the denominator as $1 - z^{-5} = (1 - z^{-1})(1 + z^{-1} + z^{-2} + z^{-3} + z^{-4})$, we get the cascaded transfer functions:

$$H(z) = \frac{z^{-1} + 2z^{-2} + 3z^{-3} + 4z^{-4}}{1 - z^{-5}} = \frac{z^{-1}}{1 - z^{-1}} \cdot \frac{1 + 2z^{-1} + 3z^{-2} + 4z^{-3}}{1 + z^{-1} + z^{-2} + z^{-3} + z^{-4}}$$

Figure P7.2 shows the cascade of these two filters, with each filter realized in its canonical form. The corresponding sample processing algorithm is as follows:

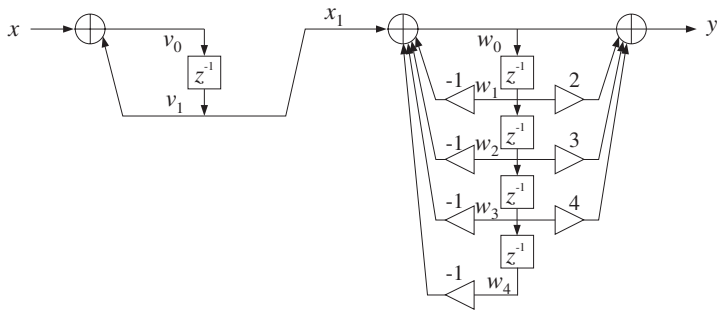


Fig. P7.2 Cascade realization of Problem 7.1.

for each input x do:
$v_0 = x + v_1$
$x_1 = v_1$
delay$(1, \mathbf{v})$
$w_0 = x_1 - w_1 - w_2 - w_3 - w_4$
$y = w_0 + 2w_1 + 3w_2 + 4w_3$
delay$(4, \mathbf{w})$

where $\mathbf{v} = [v_0, v_1]$, $\mathbf{w} = [w_0, w_1, w_2, w_3, w_4]$ are the internal state vectors for the two filter sections.

Problem 7.2

In the z-domain, the input to $H_1(z)$ is $X(z) - Y(z)$. Therefore, its output will be $H_1(z)\big(Y(z) - X(z)\big)$. The input and output of $H_2(z)$ will be then :

$$H_1(z)\big(Y(z) - X(z)\big) - Y(z) \longrightarrow H_2(z)\big[H_1(z)\big(Y(z) - X(z)\big) - Y(z)\big]$$

Adding the noise component $E(z)$ to the output of $H_2(z)$ will generate $Y(z)$, resulting in:

$$Y(z) = H_2(z)\big[H_1(z)\big(Y(z) - X(z)\big) - Y(z)\big] + E(z)$$

Moving the Y-dependent terms to the left, we get:

$$\big(1 + H_2(z) + H_1(z)H_2(z)\big)Y(z) = H_1(z)H_2(z)X(z) + E(z)$$

and solving for $Y(z)$:

$$Y(z) = \frac{H_1(z)H_2(z)}{1 + H_2(z) + H_1(z)H_2(z)}X(z) + \frac{1}{1 + H_2(z) + H_1(z)H_2(z)}E(z)$$

Thus, the desired transfer functions are identified to be:

$$H_x(z) = \frac{H_1(z)H_2(z)}{1 + H_2(z) + H_1(z)H_2(z)}$$

$$H_e(z) = \frac{1}{1 + H_2(z) + H_1(z)H_2(z)}$$

The conditions that $H_x(z)$ be a plain delay and $H_e(z)$ a double differentiator, gives the two equations:

$$\frac{H_1(z)H_2(z)}{1+H_2(z)+H_1(z)H_2(z)} = z^{-1}$$

$$\frac{1}{1+H_2(z)+H_1(z)H_2(z)} = (1-z^{-1})^2$$

which can be solved for $H_1(z)$ and $H_2(z)$ giving:

$$H_1(z) = \frac{1}{1-z^{-1}}$$

$$H_2(z) = \frac{z^{-1}}{1-z^{-1}}$$

They are both integrators. The presence of z^{-1} in the numerator of $H_2(z)$ is necessary to make the overall closed loop computable. See also Problems 12.24 and 12.25.

Problem 7.3

Multiplying the numerator factors, we get the transfer function:

$$H(z) = \frac{z^{-1}(1+2z^{-2})(1+3z^{-2})}{1-z^{-6}} = \frac{z^{-1}+5z^{-3}+6z^{-5}}{1-z^{-6}}$$

The direct and canonical realizations are shown in Fig. P7.3.

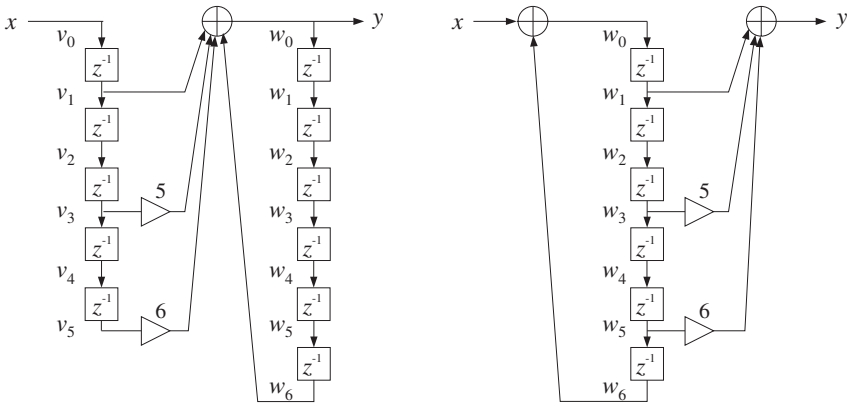


Fig. P7.3 Direct and canonical realizations of Problem 7.3.

The sample processing algorithms are expressed with the help of the routine *delay*, which updates the filter's internal state vectors.

for each input x do:
$v_0 = x$
$w_0 = v_1 + 5v_3 + 6v_5 + w_6$
$y = w_0$
delay$(5, \mathbf{v})$
delay$(6, \mathbf{w})$

for each input x do:
$w_0 = x + w_6$
$y = w_1 + 5w_3 + 6w_5$
delay$(6, \mathbf{w})$

where $\mathbf{v} = [v_0, v_1, v_2, v_4, v_5]$, $\mathbf{w} = [w_0, w_1, w_2, w_3.w_4, w_5, w_6]$ are the 6- and 7-dimensional state vectors needed for the numerator and denominator polynomials. The calls to the *delay* routines update the delay line, e.g., the call *delay*$(6, \mathbf{w})$ is equivalent to the shifts: $w_6 = w_5$, $w_5 = w_4$, $w_4 = w_3$, $w_3 = w_2$, $w_2 = w_1$, $w_1 = w_0$. Factoring the denominator as

$$1 - z^{-6} = (1-z^{-2})(1+z^{-2}+z^{-4}) = (1-z^{-2})(1+z^{-1}+z^{-2})(1-z^{-1}+z^{-2})$$

we get the cascade transfer function:

$$H(z) = \frac{z^{-1}}{1-z^{-2}} \cdot \frac{1+2z^{-2}}{1+z^{-1}+z^{-2}} \cdot \frac{1+3z^{-2}}{1-z^{-1}+z^{-2}}$$

Figure P7.4 shows the cascade of these three filters, with each filter realized in its canonical form.

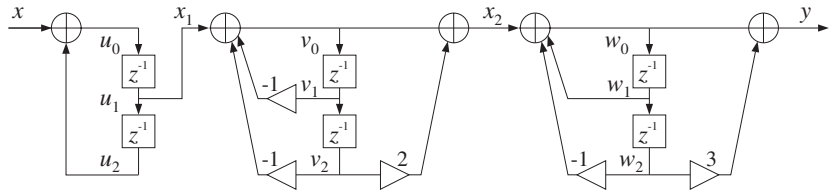


Fig. P7.4 Cascade realization of Problem 7.3.

The corresponding sample processing algorithm is as follows:

for each input x do:
$u_0 = x + u_2$
$x_1 = u_1$
delay$(2, \mathbf{u})$
$v_0 = x_1 - v_1 - v_2$
$x_2 = v_0 + 2v_2$
delay$(2, \mathbf{v})$
$w_0 = x_2 + w_1 - w_2$
$y = w_0 + 3w_2$
delay$(2, \mathbf{w})$

where $\mathbf{u} = [u_0, u_1, u_2]$, $\mathbf{v} = [v_0, v_1, v_2]$, $\mathbf{w} = [w_0, w_1, w_2]$, are the internal state vectors for the three filter sections.

To determine the impulse response, we note that the numerator corresponds to the length-6 signal $\mathbf{b} = [0, 1, 0, 5, 0, 6]$. Expanding the denominator in powers of z^{-6} will replicate the numerator periodically with period 6. But because the numerator itself has length 6, the replicas will not overlap, and we will get the repetition of the numerator coefficients:

$$\mathbf{h} = [\underbrace{0,1,0,3,0,5}, \underbrace{0,1,0,3,0,5}, \underbrace{0,1,0,3,0,5}, \ldots]$$

Problem 7.4

The z-domain versions of the time domain difference equations are:

$$\begin{aligned} v(n) &= x(n) + v(n-1) \\ y(n) &= v(n) + v(n-2) + v(n-4) \end{aligned} \quad \Rightarrow \quad \begin{aligned} V(z) &= X(z) + z^{-1}V(z) \\ Y(z) &= V(z) + z^{-2}V(z) + z^{-4}V(z) \end{aligned}$$

Eliminating $V(z)$ in favor of $Y(z), X(z)$, gives:

$$V(z) = \frac{X(z)}{1 - z^{-1}}, \qquad Y(z) = (1 + z^{-2} + z^{-4})V(z) = \frac{1 + z^{-2} + z^{-4}}{1 - z^{-1}}X(z)$$

Therefore, the transfer function is:

$$H(z) = \frac{1 + z^{-2} + z^{-4}}{1 - z^{-1}} = \frac{1 + z^{-1} + z^{-2}}{1 - z^{-1}} \cdot (1 - z^{-1} + z^{-2})$$

where we factored the numerator as

$$1 + z^{-2} + z^{-4} = (1 + z^{-1} + z^{-2})(1 - z^{-1} + z^{-2})$$

The direct, canonical, and cascade realizations are shown in Fig. P7.5.

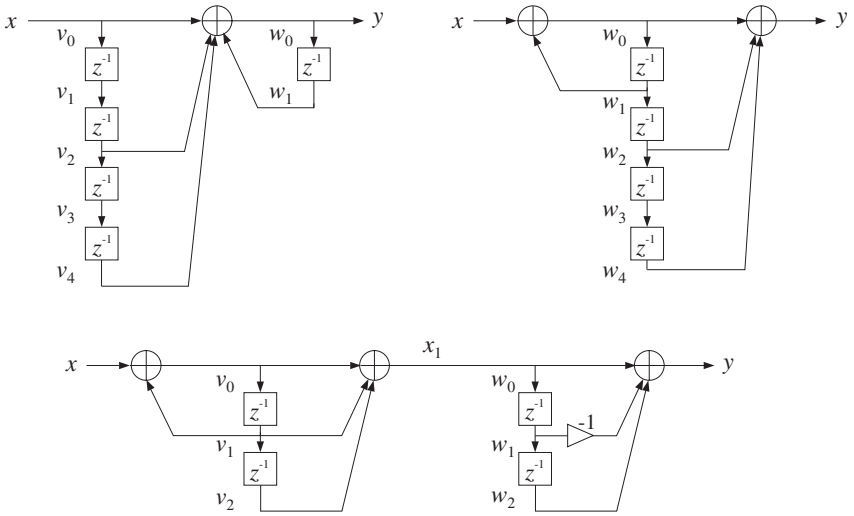


Fig. P7.5 Direct, canonical, and cascade realizations of Problem 7.4.

The direct and canonical sample processing algorithms are:

for each input x do:
$v_0 = x$
$w_0 = v_0 + v_2 + v_4 + w_1$
$y = w_0$
delay$(4, \mathbf{v})$
delay$(1, \mathbf{w})$

for each input x do:
$w_0 = x + w_1$
$y = w_0 + w_2 + w_4$
delay$(4, \mathbf{w})$

And for the cascade realization:

for each input x do:
$v_0 = x + v_1$
$x_1 = v_0 + v_1 + v_2$
delay$(2, \mathbf{v})$
$w_0 = x_1$
$y = w_0 - w_1 + w_2$
delay$(2, \mathbf{w})$

Problem 7.5

Factoring the denominator, we may write the transfer function as:

$$H(z) = \frac{2 - 3z^{-1}}{1 - 0.5z^{-2}} \cdot \frac{1 + z^{-2}}{1 + 0.5z^{-2}} = \frac{2 - 3z^{-1} + 2z^{-2} - 3z^{-3}}{1 - 0.25z^{-4}}$$

The direct and canonical realizations are shown in Fig. P7.6. Their sample processing algorithms are:

for each input x do:
$v_0 = x$
$w_0 = 2v_0 - 3v_1 + 2v_2 - 3v_3 + 0.25w_4$
$y = w_0$
delay$(3, \mathbf{v})$
delay$(4, \mathbf{w})$

for each input x do:
$w_0 = x + 0.25w_4$
$y = 2w_0 - 3w_1 + 2w_2 - 3w_3$
delay$(4, \mathbf{w})$

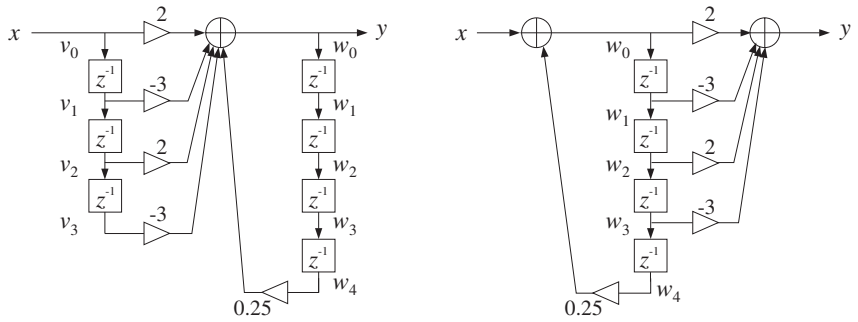


Fig. P7.6 Direct and canonical realizations of Problem 7.5.

The cascade realization is shown in Fig. P7.7. Its sample processing algorithm is:

for each input x do:
$v_0 = x + 0.5v_2$
$x_1 = 2v_0 - 3v_1$
delay$(2, \mathbf{v})$
$w_0 = x_1 - 0.5w_2$
$y = w_0 + w_2$
delay$(2, \mathbf{w})$

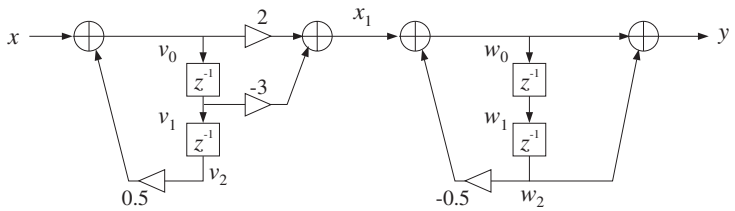


Fig. P7.7 Cascade realization of Problem 7.5.

Problem 7.6

The direct and canonical realizations are shown in Fig. P7.8. Their sample processing algorithms are:

for each input x do:
$v_0 = x$
$w_0 = v_0 - 2.25v_2 + 0.0625w_4$
$y = w_0$
delay$(2, \mathbf{v})$
delay$(4, \mathbf{w})$

for each input x do:
$w_0 = x + 0.0625w_4$
$y = w_0 - 2.25w_2$
delay$(4, \mathbf{w})$

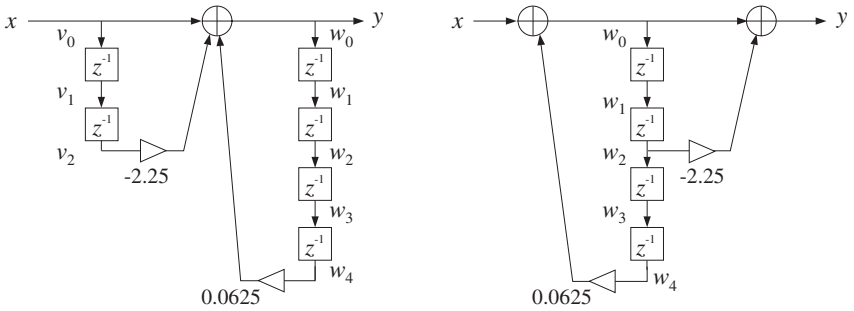


Fig. P7.8 Direct and canonical realizations of Problem 7.6.

The cascade realization is shown in Fig. P7.9. Its sample processing algorithm is:

for each input x do:
$v_0 = x - 0.25v_2$
$x_1 = v_0 - 2.25v_2$
delay$(2, \mathbf{v})$
$y = w_0 = x_1 + 0.25w_2$
delay$(2, \mathbf{w})$

Expanding $H(z)$ in partial fractions, we have:

$$H(z) = \frac{2.5}{1 - 0.5jz^{-1}} + \frac{2.5}{1 + 0.5jz^{-1}} + \frac{2}{1 - 0.5z^{-1}} + \frac{2}{1 + 0.5z^{-1}}$$

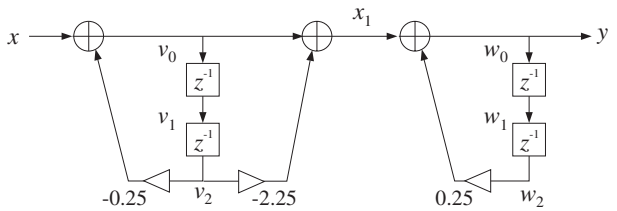


Fig. P7.9 Cascade realization of Problem 7.6.

The four poles $z = \pm 0.5, \pm 0.5j$, define only two ROCs, namely, the causal case $|z| > 0.5$ and the anticausal one $|z| < 0.5$. For example, in the causal case, the impulse response will be:

$$h(n) = 2.5(0.5j)^n u(n) + 2.5(-0.5j)^n u(n) + 2(0.5)^n u(n) + 2(-0.5)^n u(n)$$

Writing $j^n + (-j)^n = e^{j\pi n/2} + e^{-j\pi n/2} = 2\cos(\pi n/2)$, we have:

$$h(n) = 5(0.5)^n \cos(2\pi n/2) + 2(0.5)^n u(n) + 2(-0.5)^n u(n)$$

Problem 7.7

The direct and canonical realizations are shown in Fig. P7.10. Their sample processing algorithms are:

for each input x do:
$v_0 = x$
$y = w_0 = v_0 - 2v_2 + v_4 + 0.4096w_4$
delay$(4, \mathbf{v})$
delay$(4, \mathbf{w})$

for each input x do:
$w_0 = x + 0.4096w_4$
$y = w_0 - 2w_2 + w_4$
delay$(4, \mathbf{w})$

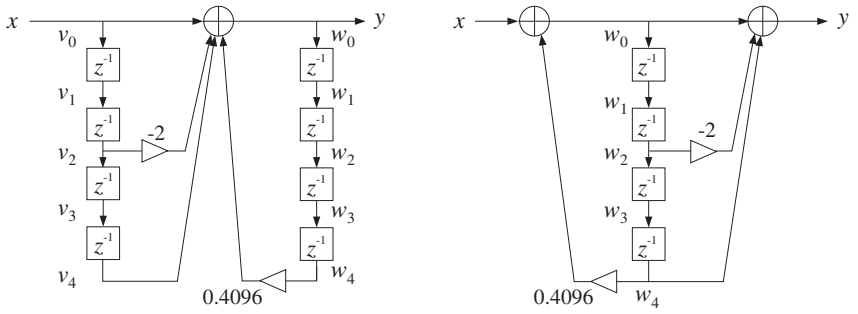


Fig. P7.10 Direct and canonical realizations of Problem 7.7.

The cascade realization is shown in Fig. P7.11. It corresponds the factorization:

$$H(z) = \frac{1 - 2z^{-2} + z^{-4}}{1 - 0.4096z^{-4}} = \frac{1 - z^{-2}}{1 - 0.64z^{-2}} \cdot \frac{1 - z^{-2}}{1 + 0.64z^{-2}}$$

Note that the numerator could also have been factored as:

$$1 - 2z^{-2} + z^{-4} = (1 - z^{-2})^2 = (1 - z^{-1})^2(1 + z^{-1})^2 = (1 - 2z^{-1} + z^{-2})(1 + 2z^{-1} + z^{-2})$$

But the chosen one has the simplest coefficients. Its sample processing algorithm is:

for each input x do:
$v_0 = x + 0.64v_2$
$x_1 = v_0 - v_2$
delay$(2, \mathbf{v})$
$w_0 = x_1 - 0.64w_2$
$y = w_0 - w_2$
delay$(2, \mathbf{w})$

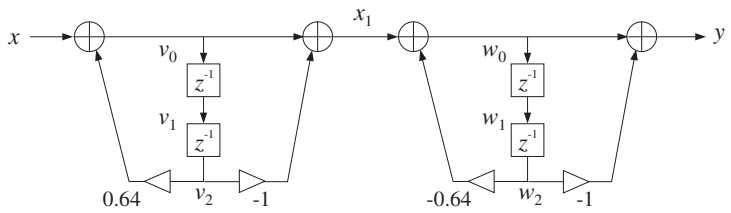


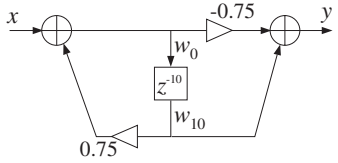
Fig. P7.11 Cascade realization of Problem 7.7.

Problem 7.8

The given difference equations describe the canonical realization form of the filter:

$$H(z) = \frac{-0.75 + z^{-10}}{1 - 0.75z^{-10}}$$

Its block diagram realization and sample processing algorithm are given below:



for each input sample x do:
$w_0 = x + 0.75w_{10}$
$y = -0.75w_0 + w_{10}$
delay$(10, \mathbf{w})$

The C version of the algorithm is as follows:

```
double reverb(x, w)
double x, w[11];
{
        int i;
        double y;

        w[0] = x + 0.75 * w[10];
        y = - 0.75 * w[0] + w[10];
```



```
        for (i=10; i>=1; i--)
                w[i] = w[i-1];

        return y;
}
```

From the transfer function, we obtain the frequency response:

$$H(\omega) = \frac{-0.75 + e^{-10j\omega}}{1 - 0.75e^{-10j\omega}} = \frac{1 - 0.75e^{10j\omega}}{1 - 0.75e^{-10j\omega}} e^{-10j\omega} = \frac{(1 - 0.75e^{-10j\omega})^*}{(1 - 0.75e^{-10j\omega})} e^{-10j\omega}$$

Noting that the ratio of two conjugate complex numbers has unit magnitude, it follows that $H(\omega)$ will have unit magnitude.

Problem 7.9

The given sample processing algorithm corresponds to the canonical realization of the filter:

$$H(z) = \frac{1 + z^{-3}}{1 - 0.64z^{-4}}$$

Factoring the numerator in the form:

$$1 + z^{-4} = (1 + z^{-1})(1 - z^{-1} + z^{-2})$$

we obtain the cascade factors:

$$H(z) = \frac{1 + z^{-3}}{1 - 0.64z^{-4}} = \frac{1 + z^{-1}}{1 - 0.8z^{-2}} \cdot \frac{1 - z^{-1} + z^{-2}}{1 + 0.8z^{-2}}$$

The cascade realization is shown in Fig. P7.12. Its sample processing algorithm is:

for each input x do:
$v_0 = x + 0.8v_2$
$x_1 = v_0 + v_1$
delay$(2, \mathbf{v})$
$w_0 = x_1 - 0.8w_2$
$y = w_0 - w_1 + w_2$
delay$(2, \mathbf{w})$

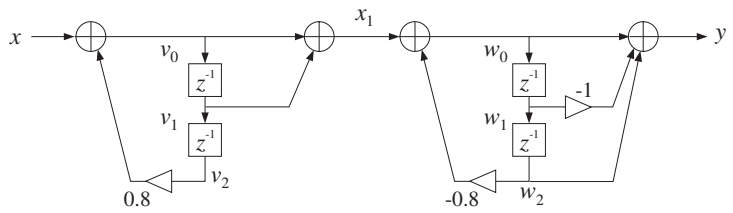


Fig. P7.12 Cascade realization of Problem 7.9.

Problem 7.10

The first filter is an FIR filter with transfer function and impulse response:

$$H(z) = (1 + z^{-2})^3 = 1 + 3z^{-2} + 3z^{-4} + z^{-6} \quad \Rightarrow \quad \mathbf{h} = [1, 0, 3, 0, 3, 0, 1]$$

The causal impulse response of the third filter is obtained by the remove restore method, that is,

$$h(n) = w(n) - w(n-4), \qquad \text{where} \qquad w(n) = (0.9)^n u(n)]$$

The first filter has zeros at $z = \pm j$, each being a triple zero. The second filter has poles at $z = \pm 0.9j$, and the third filter has zeros at the fourth roots of unity, i.e., $z = \pm 1, \pm j$, and a pole at $z = 0.9$. The pole/zero plots of the three filters are shown in Fig. P7.13 together with the corresponding sketches of the magnitude responses $|H(\omega)|$ over the right half of the Nyquist interval $0 \le \omega \le \pi$.

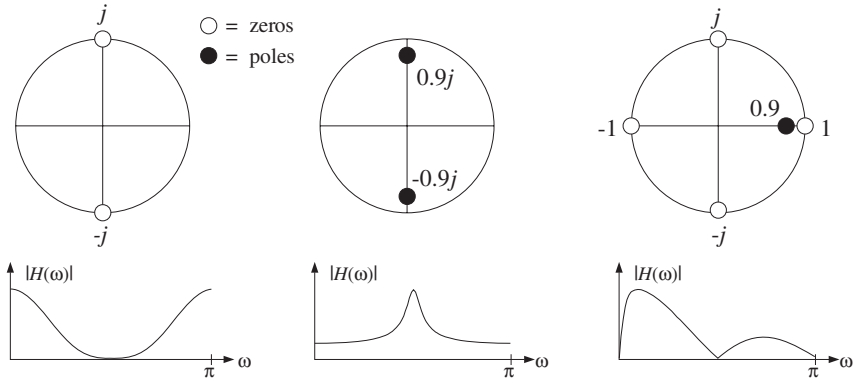


Fig. P7.13 Pole/zero patterns of Problem 7.10.

Problem 7.11

Multiplying the transfer function factors, we obtain:

$$H(z) = \frac{(1 - \sqrt{2}z^{-1} + z^{-2})(1 + \sqrt{2}z^{-1} + z^{-2})}{(1 + 0.81z^{-2})(1 - 0.81z^{-2})} = \frac{1 + z^{-4}}{1 - 0.6561z^{-4}}$$

Therefore, the zeros will be at the roots of $1 + z^{-4} = 0$, that is,

$$z^4 = -1 \qquad \Rightarrow \qquad z = e^{j\pi(2k+1)/4}, \qquad k = 0, 1, 2, 3$$

that is, the 4th roots of unity rotated by 45°. The poles are at $z = \pm 0.9m, \pm 0.9j$. Fig. P7.14 shows the pole/zero locations and the corresponding magnitude response.
The direct and canonical realizations are shown in Fig. P7.15. Their sample processing algorithms are:

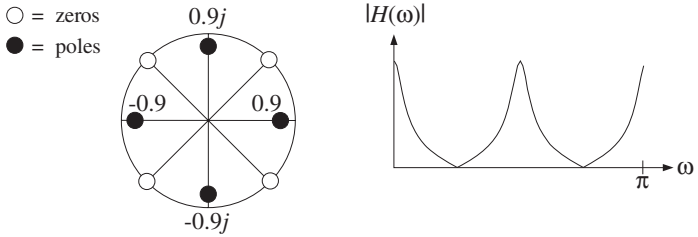


Fig. P7.14 Pole/zero plot of Problem 7.11.

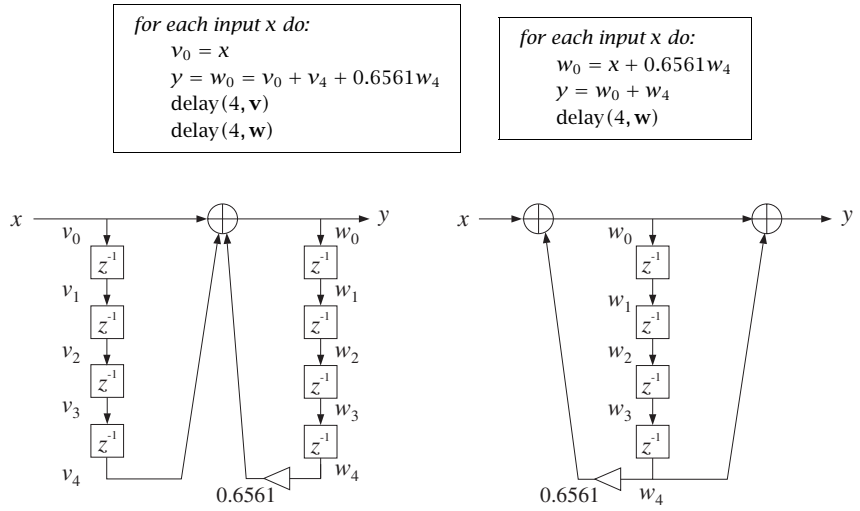


Fig. P7.15 Direct and canonical realizations of Problem 7.11.

The cascade realization is shown in Fig. P7.16. Its sample processing algorithm is:

for each input x do:
$v_0 = x - 0.81v_2$
$x_1 = v_0 - \sqrt{2}v_1 + v_2$
delay(2, **v**)
$w_0 = x_1 + 0.81w_2$
$y = w_0 + \sqrt{2}w_1 + w_2$
delay(2, **w**)

Problem 7.12

The zeros are at the roots of $1 + 16z^{-4} = 0$, that is,

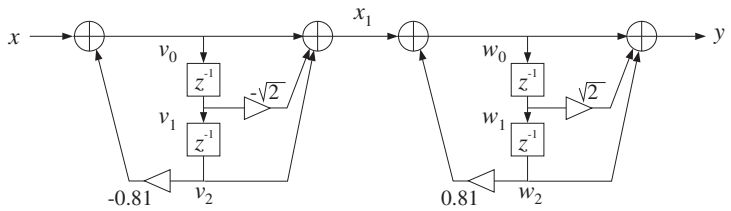


Fig. P7.16 Cascade realization of Problem 7.11.

$$z^4 = -16 \qquad \Rightarrow \qquad z = 2e^{j\pi(2k+1)/4}, \qquad k = 0,1,2,3$$

that is, the 4th roots of unity rotated by 45° and scaled in magnitude by a factor of 2. Similarly, the poles are at:

$$z^4 = -\frac{1}{16} \qquad \Rightarrow \qquad z = 0.5e^{j\pi(2k+1)/4}, \qquad k = 0,1,2,3$$

Fig. P7.17 shows the pole/zero locations.

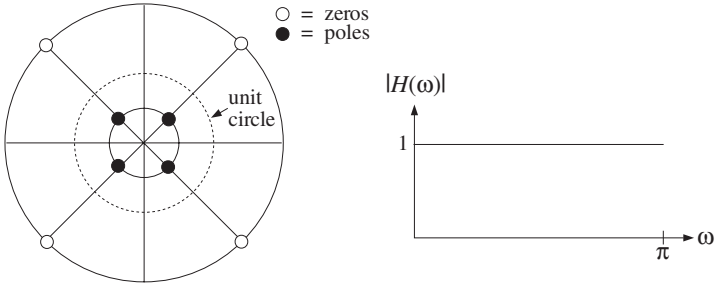


Fig. P7.17 Pole/zero plot of Problem 7.12.

The magnitude response is constant in ω. Indeed, setting $c = 1/16$, we have:

$$H(\omega) = \frac{c + e^{-4j\omega}}{1 + ce^{-4j\omega}} = \frac{e^{-4j\omega}(1 + ce^{4j\omega})}{1 + ce^{-4j\omega}} = e^{-4j\omega}\frac{(1 + ce^{-4j\omega})^*}{(1 + ce^{-4j\omega})}$$

The ratio of the two conjugate factors will have unity magnitude, giving

$$|H(\omega)| = |e^{-4j\omega}| \cdot \left|\frac{(1 + ce^{-4j\omega})^*}{(1 + ce^{-4j\omega})}\right| = 1 \cdot 1 = 1$$

To get the cascade form, we may factor the numerator and denominator using the identity:

$$1 + a^4z^{-4} = (1 - \sqrt{2}az^{-1} + a^2z^{-2})(1 + \sqrt{2}az^{-1} + a^2z^{-2})$$

Thus the numerator becomes:

$$\frac{1}{16} + z^{-4} = \frac{1}{16}(1 + 2^4z^{-4}) = \frac{1}{16}(1 - 2\sqrt{2}z^{-1} + 4z^{-2})(1 + 2\sqrt{2}z^{-1} + 4z^{-2})$$

Noting that $1/16 = 0.0625$, the transfer function is then:

$$H(z) = \frac{0.0625 + z^{-4}}{1 + 0.0625z^{-4}} = 0.0625 \cdot \frac{1 - 2\sqrt{2}z^{-1} + 4z^{-2}}{1 - 0.5\sqrt{2}z^{-1} + 0.25z^{-2}} \cdot \frac{1 + 2\sqrt{2}z^{-1} + 4z^{-2}}{1 + 0.5\sqrt{2}z^{-1} + 0.25z^{-2}}$$

The direct and canonical realizations are shown in Fig. P7.18. Their sample processing algorithms are:

for each input x do:
$v_0 = x$
$y = w_0 = 0.0625v_0 + v_4 - 0.0625w_4$
delay$(4, \mathbf{v})$
delay$(4, \mathbf{w})$

for each input x do:
$w_0 = x - 0.0625w_4$
$y = 0.0625w_0 + w_4$
delay$(4, \mathbf{w})$

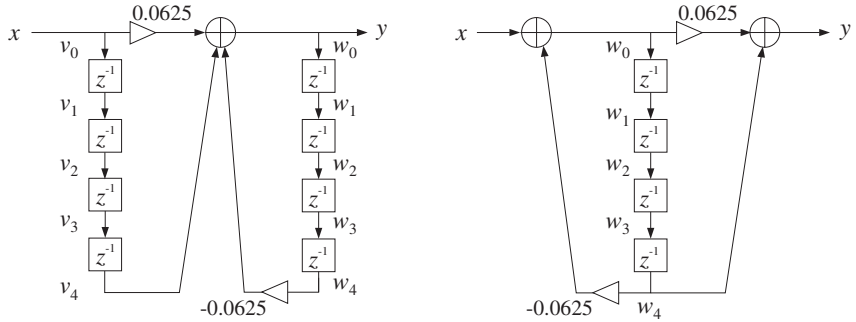


Fig. P7.18 Direct and canonical realizations of Problem 7.12.

The cascade realization is shown in Fig. P7.19. Its sample processing algorithm is:

for each input x do:
$v_0 = 0.0625x + 0.5\sqrt{2}v_1 - 0.25v_2$
$x_1 = v_0 - 2\sqrt{2}v_1 + 4v_2$
delay$(2, \mathbf{v})$
$w_0 = x_1 - 0.5\sqrt{2}w_1 - 0.25w_2$
$y = w_0 + 2\sqrt{2}w_1 + 4w_2$
delay$(2, \mathbf{w})$

Finally, the impulse response may be obtained by writing:

$$H(z) = \frac{c + z^{-4}}{1 + cz^{-4}} = c + \frac{(1 - c^2)z^{-4}}{1 + cz^{-4}} = c + (1 - c^2)[z^{-4} - cz^{-8} + c^2z^{-12} - c^3z^{-16} + \cdots]$$

Extracting the coefficients of z^{-1}, we find:

$$h(n) = \begin{cases} c & \text{if } n = 0 \\ -(1 - c^2)c^{(n-4)/4} & \text{if } n \text{ is an even multiple of 4} \\ (1 - c^2)c^{(n-4)/4} & \text{if } n \text{ is an odd multiple of 4} \\ 0 & \text{otherwise} \end{cases}$$

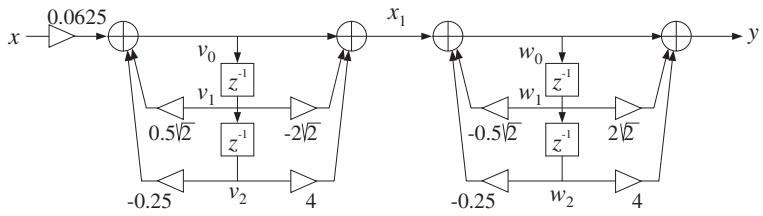


Fig. P7.19 Cascade realization of Problem 7.12.

Problem 7.13

The following program implements this computer experiment:

```
/* iirfilt.c - IIR filtering experiments */

#include <stdio.h>
#include <stdlib.h>

void cas2can();
double dir(), can(), cas();

void main()
{
    int i, j, K, M;
    double x, ydir, ycan, ycas;
    double *vdir, *wdir, *wcan, **W;
    double **A, **B, *a, *b;

    FILE *fpx;

    fpx = fopen("x.dat", "r");              /* input data */

    printf("no. sections K = ");
    scanf("%d", &K);

    M=2*K;

    wdir = (double *) calloc(M+1, sizeof(double));
    vdir = (double *) calloc(M+1, sizeof(double));
    wcan = (double *) calloc(M+1, sizeof(double));

    a = (double *) calloc(M+1, sizeof(double));
    b = (double *) calloc(M+1, sizeof(double));

    A = (double **) calloc(K, sizeof(double *));
    B = (double **) calloc(K, sizeof(double *));
    W = (double **) calloc(K, sizeof(double *));
    for (i=0; i<K; i++) {
        A[i] = (double *) calloc(3, sizeof(double));
        B[i] = (double *) calloc(3, sizeof(double));
        W[i] = (double *) calloc(3, sizeof(double));
        }

    for(i=0; i<K; i++) {
```



```
        printf("enter A[%d] = ", i);
        for(j=0; j<3; j++)
            scanf("%lf", A[i]+j);
        }

    for(i=0; i<K; i++) {
        printf("enter B[%d] = ", i);
        for(j=0; j<3; j++)
            scanf("%lf", B[i]+j);
        }

    cas2can(K, A, a);
    cas2can(K, B, b);

    printf("\n\n");

    while(fscanf(fpx, "%lf", &x) != EOF) {
        ydir = dir(M, a, M, b, vdir, wdir, x);
        ycan = can(M, a, M, b, wcan, x);
        ycas = cas(K, A, B, W, x);
        printf("% .6f   % .6f   % .6f    % .6f\n", x, ydir, ycan, ycas);
        }

    printf("\n a, b = \n");          /* direct form coefficients */
    for (i=0;i<=M;i++)
        printf("% .6lf  % .6lf\n", a[i], b[i]);
}
```

The program must be linked with the routines `cas2can`, `dir`, `can`, `cas`, and also `conv` required by `cas2can` and `sos` required by `cas`. The number of sections K and the corresponding rows of the matrices A and B are entered interactively. For example, the user input for the given transfer functions will be (starting with $K = 4$):

```
4

1      -0.373      0
1      -1.122      0.712
1      -0.891      0.360
1      -0.780      0.190

0.313   0.313      0
0.147   0.294      0.147
0.117   0.234      0.117
0.103   0.206      0.103
```

These inputs may also be saved in a file, say, `input.dat`, and piped into the program by

```
iirfilt < input.dat
```

The input signal x_n to be filtered is assumed to be in the file `x.dat`. The program stops filtering as soon as the end-of-file of `x.dat` is encountered. The first 10 of the input and output samples are shown below:

$x(n)$	$y_{\text{dir}}(n)$	$y_{\text{can}}(n)$	$y_{\text{cas}}(n)$
1.000000	0.000554	0.000554	0.000554
1.000000	0.006191	0.006191	0.006191
1.000000	0.032979	0.032979	0.032979
1.000000	0.112201	0.112201	0.112201
1.000000	0.275606	0.275606	0.275606
1.000000	0.524036	0.524036	0.524036
1.000000	0.807624	0.807624	0.807624
1.000000	1.043772	1.043772	1.043772
1.000000	1.164600	1.164600	1.164600
1.000000	1.157457	1.157457	1.157457

The outputs computed by `dir`, `can`, and `cas` are of course the same. Fig. P7.20 shows the full output, plotted together with the input. The impulse response $h(n), 0 \le n < 50$ can be computed by running this program on a file `x.dat` containing 1 followed by 49 zeros. It is shown also in Fig. P7.20. The first 10 impulse response samples computed by the three realizations are shown below:

$x(n)$	$h_{\text{dir}}(n)$	$h_{\text{can}}(n)$	$h_{\text{cas}}(n)$
1.000000	0.000554	0.000554	0.000554
0.000000	0.005637	0.005637	0.005637
0.000000	0.026788	0.026788	0.026788
0.000000	0.079222	0.079222	0.079222
0.000000	0.163405	0.163405	0.163405
0.000000	0.248430	0.248430	0.248430
0.000000	0.283588	0.283588	0.283588
0.000000	0.236148	0.236148	0.236148
0.000000	0.120828	0.120828	0.120828
0.000000	−0.007142	−0.007142	−0.007142

The given filter is an example of a lowpass Butterworth filter and the normalization gains in each section have been chosen to give unity gain at DC. Therefore, the steady-state DC response will be

$$H(\omega)\,|_{\omega=0} = H(z)\,|_{z=1} = 1$$

This can be observed in the output which rises up to 1 before it dies out to zero. Calling `cas2can` on the matrices A and B gives the direct form denominator and numerator coefficients:

a	**b**
1.000000	0.000554
−3.166000	0.003881
4.873631	0.011644
−4.465987	0.019407
2.592519	0.019407
−0.941724	0.011644
0.196860	0.003881
−0.018165	0.000554
0.000000	0.000000

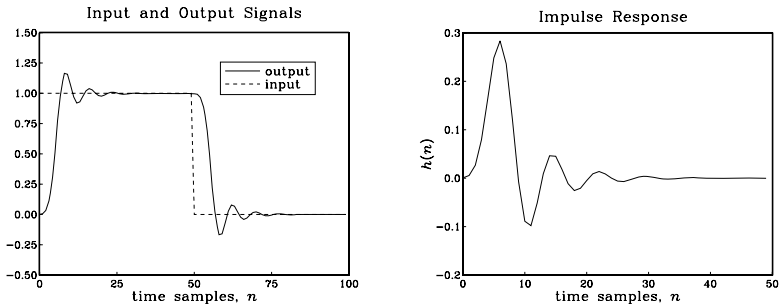


Fig. P7.20 Input, output, and impulse response of Problem 7.13.

Problem 7.14

An example of such program is given below. It is patterned after `firfilt.c` of Problem 4.10. The program reads and allocates the filter coefficient arrays. The last `while` loop performs the actual filtering:

```
/* canfilt.c - IIR filtering in canonical form */

#include <stdlib.h>
#include <stdio.h>

#define MAX 64                                  initial allocation size

double can();

void main(int argc, char **argv)
{
    FILE *fpa, *fpb;                            coefficient files
    double *a, *b, *w, x, y;                    coeffs., state, input, output
    int L, M, K, i;
    int max = MAX, dmax = MAX;                  initial allocation and increment

    if (argc != 3) {
        fprintf(stderr, "Usage: canfilt a.dat b.dat <x.dat >y.dat\n");
        exit(0);
        }

    if ((fpa = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "Can't open filter file: %s\n", argv[1]);
        exit(0);
        }

    if ((fpb = fopen(argv[2], "r")) == NULL) {
        fprintf(stderr, "Can't open filter file: %s\n", argv[2]);
        exit(0);
        }

    a = (double *) calloc(max + 1, sizeof(double));
    for (M=0;; M++) {
        if (M == max) {
```

```
            max += dmax;
            a = (double *) realloc((char *) a, (max + 1) * sizeof(double));
            }
        if (fscanf(fpa, "%lf", a + M) == EOF) break;
        }
    M--;
    a = (double *) realloc((char *) a, (M + 1) * sizeof(double));

    b = (double *) calloc(max + 1, sizeof(double));
    for (L=0;; L++) {
        if (L == max) {
            max += dmax;
            b = (double *) realloc((char *) b, (max + 1) * sizeof(double));
            }
        if (fscanf(fpb, "%lf", b + L) == EOF) break;
        }
    L--;
    b = (double *) realloc((char *) b, (L + 1) * sizeof(double));

    K = (L <= M) ? M : L;

    w = (double *) calloc((K + 1), sizeof(double));

    while(scanf("%lf", &x) != EOF) {          process input samples
        y = can(M, a, L, b, w, x);
        printf("%lf\n", y);
        }
}
```

Problem 7.15

An example of such program is given below. The program reads and allocates the filter coefficient matrices A and B. To facilitate the initial allocation of A, a temporary array **a** is used to read the rows of `A.dat` in a concatenated form, and then it reshapes it into the $K\times 3$ matrix A. The last `while` loop performs the actual filtering:

```
/* casfilt.c - IIR filtering in cascade form */

#include <stdlib.h>
#include <stdio.h>

#define MAX 64                                    initial allocation size

double cas();

void main(int argc, char **argv)
{
    FILE *fpa, *fpb;                              coefficient files
    double **A, **B, **W, x, y;
    double *a;                                    temporary coefficient array
    int M, K, i, j, m;
    int max = MAX, dmax = MAX;                    allocation and increment

    if (argc != 3) {
        fprintf(stderr, "Usage: casfilt A.dat B.dat <x.dat >y.dat\n");
        exit(0);
        }
```



```
    if ((fpa = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "Can't open filter file: %s\n", argv[1]);
        exit(0);
        }

    if ((fpb = fopen(argv[2], "r")) == NULL) {
        fprintf(stderr, "Can't open filter file: %s\n", argv[2]);
        exit(0);
        }

    a = (double *) calloc(max + 1, sizeof(double));
    for (M=0;; M++) {
        if (M == max) {
            max += dmax;
            a = (double *) realloc((char *) a, (max + 1) * sizeof(double));
            }
        if (fscanf(fpa, "%lf", a + M) == EOF) break;
        }
    a = (double *) realloc((char *) a, (M + 1) * sizeof(double));

    if (M%3 != 0) {
        fprintf(stderr, "all rows of A must be 3-dimensional");
        exit(0);
        }
    else
       K = M / 3;

    A    = (double **) calloc(K, sizeof(double *));
    B    = (double **) calloc(K, sizeof(double *));
    W    = (double **) calloc(K, sizeof(double *));
    for (i=0; i<K; i++) {
       A[i] = (double *) calloc(3, sizeof(double));
       B[i] = (double *) calloc(3, sizeof(double));
       W[i] = (double *) calloc(3, sizeof(double));
       }

    for(i=0; i<K; i++)
       for(j=0; j<3; j++)
           A[i][j] = a[3*i + j];

    for(i=0; i<K; i++)
       for(j=0; j<3; j++)
           fscanf(fpb, "%lf", B[i] + j);

    while(scanf("%lf", &x) != EOF) {          process input samples
        y = cas(K, A, B, W, x);
        printf("%lf\n", y);
        }
}
```

The MATLAB version `casfilt.m` is given below. It calls `cas.m` N times, where N is the input length:

```
% casfilt.m - IIR filtering in cascade form
%
% y = casfilt(B, A, x);
%
```

```
% B = Kx3 numerator matrix   (K = number of sections)
% A = Kx3 denominator matrix
% x = row vector input
% y = row vector output

function y = casfilt(B, A, x)

[K, K1] = size(A);
[N1, N] = size(x);
W = zeros(K,3);

for n = 1:N,
       [y(n), W] = cas(K, B, A, W, x(n));
end
```

Problem 7.16

The two filters are:

$$H_1(z) = 0.70849 \cdot \frac{1+z^{-8}}{1-0.0625z^{-8}}, \qquad H_2(z) = 0.98021 \cdot \frac{1+z^{-8}}{1+0.94z^{-8}} \cdot \frac{1-0.96z^{-8}}{1-0.98z^{-8}}$$

The scale factors, were chosen such that the magnitude responses are unity at $\omega = \omega_0 = 0.5\pi/8$, that is,

$$|H_1(\omega_0)| = |H_2(\omega_0)| = 1$$

The following program implements this computer experiment. The sample processing algorithms of Examples 7.4.3 and 7.4.4 remain the same, except the input sample x gets multiplied by the corresponding normalization gains:

```
/* combex.c - comb filtering examples */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define A0  1.0
#define A1  0.5
#define A2  0.5
#define A3  0.5

#define G1  0.70849
#define G2  0.98021

void delay();

void main()
{
       int n, N;
       double s, x, y1, x1, y2;
       double *w, *v, *u;
       double pi = 4*atan(1.0), w0, w1, w2, w3;

       FILE *fps, *fpx, *fpy1, *fpy2;

       fps  = fopen("s.dat", "w");
       fpx  = fopen("x.dat", "w");
       fpy1 = fopen("y1.dat", "w");
       fpy2 = fopen("y2.dat", "w");

       w = (double *) calloc(9, sizeof(double));        /* filter 1 */
       u = (double *) calloc(9, sizeof(double));        /* filter 2, section 1 */
       v = (double *) calloc(9, sizeof(double));        /* filter 2, section 2 */

       w0 = 0.50 * pi / 8;
       w1 = 0.75 * pi / 8;
       w2 = 1.00 * pi / 8;
       w3 = 3.00 * pi / 8;

       printf("enter input length N = ");
       scanf("%d", &N);

       for (n=0; n<N; n++) {
              s = A0 * cos(w0*n) + A1 * cos(w1*n);
              x = s + A2 * cos(w2*n) +  A3 * cos(w3*n);

              w[0] = G1 * x + 0.0625 * w[8];            /* filter 1 */
              y1 = w[0] + w[8];
              delay(8, w);

              u[0] = G2 * x - 0.94 * u[8];              /* filter 2 */
              x1 = u[0] + u[8];                         /* section 1 output */
              delay(8, u);
              v[0] = x1 + 0.98 * v[8];
              y2 = v[0] - 0.96 * v[8];                  /* section 2 output */
              delay(8, v);

              fprintf(fps,  "%.8lf\n", s);
              fprintf(fpx,  "%.8lf\n", x);
              fprintf(fpy1, "%.8lf\n", y1);
              fprintf(fpy2, "%.8lf\n", y2);
              }
}
```

Fig. P7.21 shows the desired, noise-free, signal $s(n)$ and the noisy signal $x(n)$, which serves as the input to the comb filters. The corresponding output signals $y_1(n)$ and $y_2(n)$ from $H_1(z)$ and $H_2(z)$ are shown in Fig. P7.22.
The first filter has a short time constant, but does not remove the noise component well. The second filter has a longer time constant, but it removes the noise very well.

Problem 7.17

Performing a partial fraction expansion on $H_1(z)$ and $H_2(z)$, we find (where the PFE coefficients are given with 5-digit accuracy):

$$H_1(z) = -11.33584 + \frac{12.04433}{1-0.0625z^{-8}}$$

$$H_2(z) = 1.02150 - \frac{0.06191}{1+0.94z^{-8}} + \frac{0.02063}{1-0.98z^{-8}}$$

Expanding in powers of z^{-8} and extracting coefficients, we find the impulse responses, for $0 \le n \le 16$:

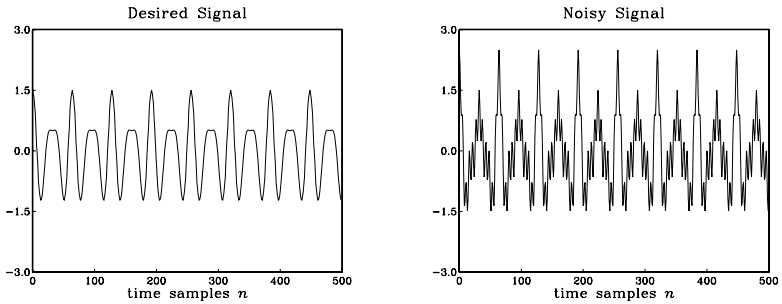


Fig. P7.21 Desired and noisy inputs of Problem 7.16.

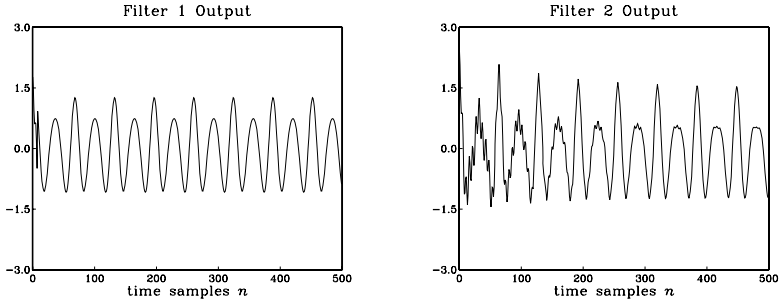


Fig. P7.22 Outputs of comb filters 1 and 2 of Problem 7.16.

n	$h_1(n)$	$h_2(n)$
0	0.70849	0.98021
1	0	0
2	0	0
3	0	0
4	0	0
5	0	0
6	0	0
7	0	0
8	0.75277	0.07841
9	0	0
10	0	0
11	0	0
12	0	0
13	0	0
14	0	0
15	0	0
16	0.04705	−0.03489

The impulse responses of the two filters can be computed by the same program as above, but with changing the input $x(n)$ into an impulse $\delta(n)$. The computed impulse responses are shown in Fig. P7.23

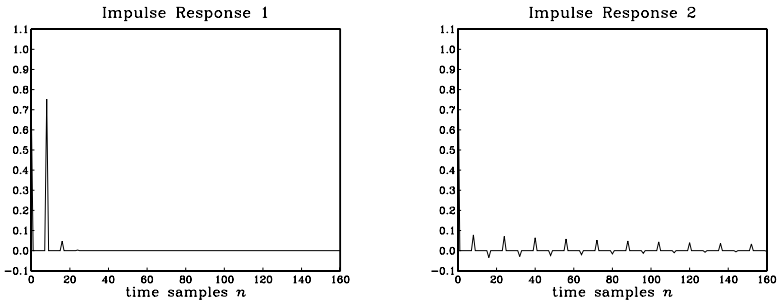


Fig. P7.23 Impulse responses of comb filters 1 and 2 of Problem 7.17.

Problem 7.18

The following program replaces `iirfilt.c` of Problem 7.13. For comparison, `dir.c` is left in the program as before.

```
/* iirfilt.c - IIR filtering experiments - circular buffer versions */

#include <stdio.h>
#include <stdlib.h>

void cas2can();
double dir(), ccan(), ccas();

void main()
{
    int i, j, K, M;
    double x, ydir, ycan, ycas;
    double *vdir, *wdir, *wcan, **W;
    double **A, **B, *a, *b;
    double *p, **P;                         /* circular pointers */

    FILE *fpx;

    fpx = fopen("x.dat", "r");

    printf("no. sections K = ");
    scanf("%d", &K);

    M=2*K;

    wdir = (double *) calloc(M+1, sizeof(double));
    vdir = (double *) calloc(M+1, sizeof(double));
    wcan = (double *) calloc(M+1, sizeof(double));
    p = wcan;                                           /* initialize */
```

```
    a = (double *) calloc(M+1, sizeof(double));
    b = (double *) calloc(M+1, sizeof(double));

    A    = (double **) calloc(K, sizeof(double *));
    B    = (double **) calloc(K, sizeof(double *));
    W    = (double **) calloc(K, sizeof(double *));
    P    = (double **) calloc(K, sizeof(double *));     /* allocate */
    for (i=0; i<K; i++) {
       A[i] = (double *) calloc(3, sizeof(double));
       B[i] = (double *) calloc(3, sizeof(double));
       W[i] = (double *) calloc(3, sizeof(double));
       P[i] = W[i];                                     /* initialize */
       }

    for(i=0; i<K; i++) {
        printf("enter A[%d] = ", i);
        for(j=0; j<3; j++)
            scanf("%lf", A[i]+j);
        }

    for(i=0; i<K; i++) {
        printf("enter B[%d] = ", i);
        for(j=0; j<3; j++)
            scanf("%lf", B[i]+j);
        }

    cas2can(K, A, a);
    cas2can(K, B, b);

    printf("\n\n");

    while(fscanf(fpx, "%lf", &x) != EOF) {
        ydir = dir(M, a, M, b, vdir, wdir, x);
        ycan = ccan(M, a, b, wcan, &p, x);
        ycas = ccas(K, A, B, W, P, x);
        printf("% .6f   % .6f   % .6f    % .6f\n", x, ydir, ycan, ycas);
        }

   printf("a, b = \n");
   for (i=0;i<=M;i++)
        printf("% .6lf  % .6lf\n", a[i], b[i]);
}
```

One minor difference in `ccan` is that it assumes the same order M for both vectors **a** and **b**; Therefore its usage is:

```
ycan = ccan(M, a, b, wcan, &p, x);
```

whereas the usage of `can` was:

```
ycan = can(M, a, M, b, wcan, x);
```

Problem 7.19

The canonical realization is shown in Fig. P7.24.
The linear and circular buffer versions of the sample processing algorithm are as follows:

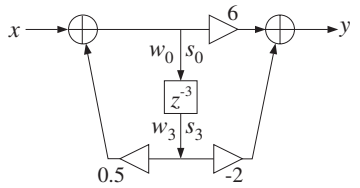


Fig. P7.24 Canonical realization of Problem 7.19.

for each input x do:
$w_0 = x + 0.5w_3$
$y = 6w_0 - 2w_3$
delay$(3, \mathbf{w})$

for each input x do:
$s_3 = \text{tap}(3, \mathbf{w}, p, 3)$
$s_0 = x + 0.5s_3$
$y = 6s_0 - 2s_3$
$*p = s_0$
cdelay$(3, \mathbf{w}, \&p)$

The impulse response can be obtained by dividing out the numerator and writing

$$H(z) = \frac{6 - 2z^{-3}}{1 - 0.5z^{-3}} = 4 + \frac{2}{1 - 0.5z^{-3}}$$

Expanding in powers of z^{-3}, we have:

$$H(z) = 4 + 2\left[1 + (0.5)z^{-3} + (0.5)^2 z^{-6} + (0.5)^3 z^{-9} + \cdots\right]$$

which gives the impulse response:

$$\mathbf{h} = [6, 0, 0, 2(0.5), 0, 0, 2(0.5)^2, 0, 0, 2(0.5)^3, 0, 0, \ldots] = [6, 0, 0, 1, 0, 0, 0.5, \ldots]$$

n	x	w_0	w_1	w_2	w_3	s_0	s_1	s_2	s_3	$y = 6s_0 - 2s_3$
0	1	1	0	0	0	1	0	0	0	6
1	0	1	0	0	0	0	1	0	0	0
2	0	1	0	0	0	0	0	1	0	0
3	0	1	0.5	0	0	0.5	0	0	1	1
4	0	0	0.5	0	0	0	0.5	0	0	0
5	0	0	0.5	0	0	0	0	0.5	0	0
6	0	0	0.5	0.5^2	0	0.5^2	0	0	0.5	0.5

The states s_i represent the delay-line outputs:

$$\mathbf{s}(n) = \begin{bmatrix} s_0(n) \\ s_1(n) \\ s_2(n) \\ s_3(n) \end{bmatrix} = \begin{bmatrix} w(n) \\ w(n-1) \\ w(n-2) \\ w(n-3) \end{bmatrix}$$

that is, they are the successively delayed versions of $w(n)$, which is the intermediate signal obtained at the output of the input adder. It has z-transform:

$$W(z) = \frac{1}{D(z)} X(z) = \frac{1}{1 - 0.5z^{-3}} X(z)$$

For a delta-function input, we have $X(z)=1$, giving:

$$W(z)=\frac{1}{1-0.5z^{-3}}=1+(0.5)z^{-3}+(0.5)^2z^{-6}+(0.5)^3z^{-9}+\cdots$$

and in the time domain:

$$w(n)=[1,0,0,0.5,0,0,0.5^2,0,0,\dots]$$

For the input $\mathbf{x}=[1,2,3]$, we have:

n	x	w_0	w_1	w_2	w_3	s_0	s_1	s_2	s_3	$y=6s_0-2s_3$
0	1	1	0	0	0	1	0	0	0	6
1	2	1	0	0	2	2	1	0	0	12
2	3	1	0	3	2	3	2	1	0	18
3	0	1	0.5	3	2	0.5	3	2	1	1
4	0	1	0.5	3	2	1	0.5	3	2	2
5	0	1	0.5	3	1.5	1.5	1	0.5	3	3
6	0	1	0.5	0.25	1.5	0.25	1.5	1	0.5	0.5

Problem 7.20

The full precision direct form denominator is:

$$\mathbf{a}=[1,a_1,a_2,a_3,a_4]=[1,-3.502,5.024,-3.464,0.979] \tag{P7.1}$$

The full precision cascade coefficients are:

$$\begin{aligned}\mathbf{a}_0&=[1,a_{01},a_{02}]=[1,-1.8955,0.9930]\\ \mathbf{a}_1&=[1,a_{11},a_{12}]=[1,-1.6065,0.9859]\end{aligned} \tag{P7.2}$$

The poles are at:

$$\begin{aligned}p_0&=R_0e^{j\omega_0}, \qquad R_0=0.9965, \quad \omega_0=0.1\pi\\ p_1&=R_1e^{j\omega_1}, \qquad R_1=0.9929, \quad \omega_1=0.2\pi\end{aligned} \tag{P7.3}$$

If we round the direct form coefficient vector $\mathbf{a}$ to 2-digit accuracy, we obtain:

$$\hat{\mathbf{a}}=[1,\hat{a}_1,\hat{a}_2,\hat{a}_3,\hat{a}_4]=[1,-3.50,5.02,-3.46,0.98] \tag{P7.4}$$

and the corresponding direct form transfer function:

$$\hat{H}_{\text{dir}}(z)=\frac{1}{1+\hat{a}_1z^{-1}+\hat{a}_2z^{-2}+\hat{a}_3z^{-3}+\hat{a}_4z^{-4}} \tag{P7.5}$$

This filter is *unstable.* Indeed, its four poles, the roots of the denominator polynomial, are:

$$\begin{aligned}\hat{p}_0&=\hat{R}_0e^{j\hat{\omega}_0}, \qquad \hat{R}_0=1.0045, \quad \hat{\omega}_0=0.1045\pi\\ \hat{p}_1&=\hat{R}_1e^{j\hat{\omega}_1}, \qquad \hat{R}_1=0.9860, \quad \hat{\omega}_1=0.1989\pi\end{aligned} \tag{P7.6}$$

and their conjugates. Thus, the p_0 pole has moved *outside* the unit circle, rendering the filter unstable. The seemingly benign replacement of $\mathbf{a}$ by the rounded $\hat{\mathbf{a}}$ has made the direct form realization unusable. The impulse responses of the two filters (7.7.1) and (P7.5) are shown in Fig. P7.25. The latter is blowing up exponentially.

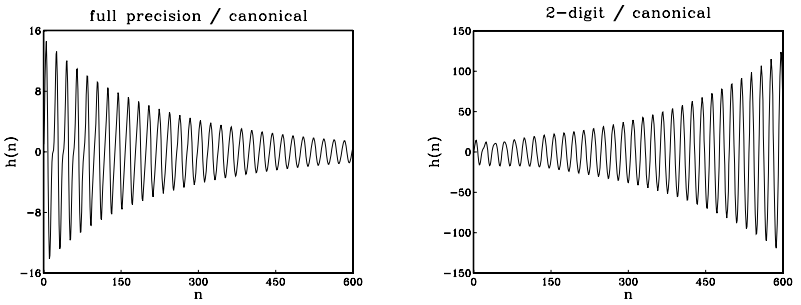


Fig. P7.25 Full and 2-digit precision filters.

The cascade realization, on the other hand, is more robust, in the sense that if we round *its* coefficients also to 2-digit accuracy, the filter will remain stable. Indeed, rounding the SOS coefficients (P7.2) to 2-digits gives:

$$\begin{aligned}\hat{\mathbf{a}}_0&=[1,\hat{a}_{01},\hat{a}_{02}]=[1,-1.90,0.99]\\ \hat{\mathbf{a}}_1&=[1,\hat{a}_{11},\hat{a}_{12}]=[1,-1.61,0.99]\end{aligned} \tag{P7.7}$$

The roots of these sections are:

$$\begin{aligned}\hat{p}_0&=\hat{R}_0e^{j\hat{\omega}_0}, \qquad \hat{R}_0=0.9950 \quad \hat{\omega}_0=0.0961\pi\\ \hat{p}_1&=\hat{R}_1e^{j\hat{\omega}_1}, \qquad \hat{R}_1=0.9950, \quad \hat{\omega}_1=0.2000\pi\end{aligned} \tag{P7.8}$$

and their conjugates. They are both inside the unit circle, therefore, the filter remains stable. The corresponding cascade transfer function will be:

$$\hat{H}_{\text{cas}}(z)=\frac{1}{1+\hat{a}_{01}z^{-1}+\hat{a}_{02}z^{-2}}\cdot\frac{1}{1+\hat{a}_{11}z^{-1}+\hat{a}_{12}z^{-2}} \tag{P7.9}$$

The filters $\hat{H}_{\text{dir}}(z)$ and $\hat{H}_{\text{cas}}(z)$ are not equal. Rounding the cascade coefficients to 2-digit accuracy is *not equivalent* to rounding the direct form coefficients, because rounding does not preserve products—the rounded product of two numbers is not equal to the product of the two rounded numbers. Therefore, rounding does not preserve convolution, that is,

$$\hat{\mathbf{a}}=\widehat{\mathbf{a}_0*\mathbf{a}_1}\neq\hat{\mathbf{a}}_0*\hat{\mathbf{a}}_1$$

Indeed, carrying out the convolution of the two vectors in Eq. (P7.7) gives the following direct form vector, which is not quite the same as that given by Eq. (P7.4):

$$\hat{\mathbf{a}}_0*\hat{\mathbf{a}}_1=[1,\ -3.5100,\ 5.0390,\ -3.4749,\ 0.9801]$$

The impulse response and the magnitude response of the cascaded filter (P7.9) are shown in Fig. P7.26, together with the exact magnitude response. The magnitude responses are plotted in dB, that is, $20\log_{10}|H(\omega)|$, over $0\le\omega\le0.5\pi$, which is the right half of the Nyquist interval. Thus, in the cascade case the rounding operation changed the magnitude response somewhat, but has not affected the stability of the filter.

The sensitivity of the direct form and the robustness of the cascade form can be understood analytically in the following way. The zeros of a polynomial are indirect functions of the coefficients

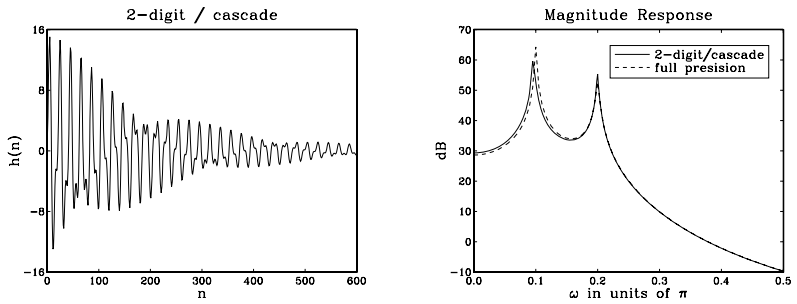


Fig. P7.26 Impulse and magnitude responses of 2-digit-precision cascade form.

of the polynomial, therefore, small changes in the latter will induce changes in the former. For the direct form, the induced changes in the zeros can be *large*, thus, causing them to move outside the unit circle. In the cascade form, the induced changes remain small. For example, consider the above fourth order denominator vector **a**. The relationship of the zeros to the coefficients is established via the polynomial identity:

$$(1-p_0z^{-1})(1-p_0^*z^{-1})(1-p_1z^{-1})(1-p_1^*z^{-1}) = 1+a_1z^{-1}+a_2z^{-2}+a_3z^{-3}+a_4z^{-4}$$

A small change in the coefficients will induce a small change in the zeros p_0 and p_1, that is,

$$\mathbf{a} \to \mathbf{a} + d\mathbf{a} \quad \Rightarrow \quad p_i \to p_i + dp_i, \quad i = 0, 1$$

To determine dp_0 and dp_1 in terms of $d\mathbf{a}$, we take differentials of both sides of the above polynomial identity to get:

$$\begin{aligned}
&-dp_0z^{-1}(1-p_0^*z^{-1})(1-p_1z^{-1})(1-p_1^*z^{-1})\\
&-(1-p_0z^{-1})dp_0^*z^{-1}(1-p_1z^{-1})(1-p_1^*z^{-1})\\
&-(1-p_0z^{-1})(1-p_0^*z^{-1})dp_1z^{-1}(1-p_1^*z^{-1})\\
&-(1-p_0z^{-1})(1-p_0^*z^{-1})(1-p_1z^{-1})dp_1^*z^{-1}\\
&\quad = da_1z^{-1}+da_2z^{-2}+da_3z^{-3}+da_4z^{-4}
\end{aligned}$$

which is still an identity in z. Setting $z = p_0$ will remove all but the first term in the left-hand-side, giving the relationship:

$$-dp_0p_0^{-1}(1-p_0^*p_0^{-1})(1-p_1p_0^{-1})(1-p_1^*p_0^{-1}) = da_1p_0^{-1}+da_2p_0^{-2}+da_3p_0^{-3}+da_4p_0^{-4}$$

Multiplying by p_0^4 and solving for dp_0, we obtain

$$dp_0 = -\frac{p_0^3da_1+p_0^2da_2+p_0da_3+da_4}{(p_0-p_0^*)(p_0-p_1)(p_0-p_1^*)} \tag{P7.10}$$

Similarly, setting $z = p_1$, we find for dp_1:

$$dp_1 = -\frac{p_1^3da_1+p_1^2da_2+p_1da_3+da_4}{(p_1-p_0^*)(p_1-p_0)(p_1-p_1^*)} \tag{P7.11}$$

Because, the denominator involves the differences between poles, that is, the distances from all the other poles, it follows that if the poles are closely clustered, the induced changes in the poles dp_i will be large even if the coefficient changes da_i are small. The above sensitivity formulas generalize easily [2,3,246] to the case of an order-M filter $\mathbf{a} = [1, a_1, a_2, \ldots, a_M]$ with M zeros $p_1, p_2, \ldots, p_M$:

$$dp_i = \frac{\partial p_i}{\partial a_1}da_1 + \frac{\partial p_i}{\partial a_2}da_2 + \cdots + \frac{\partial p_i}{\partial a_M}da_M$$

where the partial derivatives are:

$$\frac{\partial p_i}{\partial a_m} = -\frac{p_i^{M-m}}{\prod_{\substack{j=1\\ j\neq i}}^{M}(p_i-p_j)} \tag{P7.12}$$

In the cascade realization, p_0 is associated with the first SOS and p_1 with the second. The dependence of p_0 on $\mathbf{a}_0$ and p_1 on $\mathbf{a}_1$ is established through the identities

$$\begin{aligned}
(1-p_0z^{-1})(1-p_0^*z^{-1}) &= 1+a_{01}z^{-1}+a_{02}z^{-2}\\
(1-p_1z^{-1})(1-p_1^*z^{-1}) &= 1+a_{11}z^{-1}+a_{12}z^{-2}
\end{aligned}$$

Using the above procedure of taking differentials of both sides and setting $z = p_0$ in the first and $z = p_1$ in the second, we obtain the induced changes in the poles due to small changes in the coefficients of the sections:

$$dp_0 = -\frac{p_0da_{01}+da_{02}}{(p_0-p_0^*)}, \qquad dp_1 = -\frac{p_1da_{11}+da_{12}}{(p_1-p_1^*)} \tag{P7.13}$$

The sensitivity of each pole depends *only* on the SOS it belongs to, not on the other sections. In Eqs. (P7.10) and (P7.11), the sensitivity of each pole is *coupled* to the other poles, through the denominator factors of the form $(p_0 - p_1)$; if p_0 is near p_1, such factors cause large shifts in the pole locations.

Even though the sensitivity formulas (P7.10) and (P7.11) involve differential changes, we may actually apply them to the above example and estimate the pole shifts for the 2-digit case. Writing the rounded $\hat{\mathbf{a}}$ of Eq. (P7.4) in the form $\hat{\mathbf{a}} = \mathbf{a} + d\mathbf{a}$, we obtain subtracting (P7.4) and (P7.1):

$$d\mathbf{a} = \hat{\mathbf{a}} - \mathbf{a} = [0, da_1, da_2, da_3, da_4] = [0, 0.002, -0.004, 0.004, 0.001]$$

Using the numerical values given in Eq. (P7.3), we have

$$\begin{aligned}
p_0 &= 0.9965e^{j0.1\pi} = 0.9477 + j0.3079\\
p_1 &= 0.9929e^{j0.2\pi} = 0.8033 + j0.5836
\end{aligned}$$

With these values of the poles and da_i, we can calculate the pole shifts using (P7.10) and (P7.11). We find:

$$dp_0 = 0.0023 + j0.0159, \qquad dp_1 = -0.0033 - j0.0066$$

Therefore, the new pole locations will be:

$$\begin{aligned}
\hat{p}_0 &= p_0 + dp_0 = 0.9501 + j0.3238 = 1.0037e^{j0.1046\pi}\\
\hat{p}_1 &= p_1 + dp_1 = 0.7999 + j0.5770 = 0.9863e^{j0.1989\pi}
\end{aligned}$$

which compare well with the actual values of Eq. (P7.6). A similar calculation can be made based on Eq. (P7.13).

Problem 7.21

The numerical values of $\mathbf{a}$ are shown in the first column of Table P7.1. The second and third columns of this table show the *quantized* versions of $\mathbf{a}$, rounded to 5- and 4-digit accuracy.

$\mathbf{a}$	5-digit $\hat{\mathbf{a}}$	4-digit $\hat{\mathbf{a}}$	$\hat{\mathbf{a}}_{\text{cas}}$
1.00000000	1.00000000	1.00000000	1.00000000
−5.56573395	−5.56573000	−5.56570000	−5.56570000
13.05062482	13.05062000	13.05060000	13.05046030
−16.49540451	−16.49540000	−16.49540000	−16.49508213
11.84993647	11.84994000	11.84990000	11.84961490
−4.58649943	−4.58650000	−4.58650000	−4.58633537
0.74713457	0.74713000	0.74710000	0.74710016
relative error	3.2580×10^{-7}	1.6487×10^{-6}	1.9240×10^{-5}

Table P7.1 Exact, 5-digit, and 4-digit coefficients.

The fourth column of Table P7.1 shows the equivalent direct-form coefficient vector $\hat{\mathbf{a}}_{\text{cas}}$ arising from the cascade of the *quantized* second order section coefficients, rounded to 4-digit accuracy. The rounded $\mathbf{a}_i$ are:

$$\begin{aligned}\hat{\mathbf{a}}_0 &= [1,\ -1.8671,\ 0.9623]\\ \hat{\mathbf{a}}_1 &= [1,\ -1.8468,\ 0.8992]\\ \hat{\mathbf{a}}_2 &= [1,\ -1.8518,\ 0.8634]\end{aligned} \tag{P7.14}$$

The table also shows the relative error in each case, defined as the ratio

$$\text{relative error} = \frac{\|d\mathbf{a}\|}{\|\mathbf{a}\|} = \frac{\|\hat{\mathbf{a}}-\mathbf{a}\|}{\|\mathbf{a}\|}$$

where $\|\mathbf{a}\|$ denotes the length of the vector $\mathbf{a}$. As expected, the error in the 5-digit case is less than that in the 4-digit case. But, the error of the cascade case is larger. However, as we see below, the 4-digit cascade form gives an *adequate* approximation of the exact filter, whereas both the 5- and 4-digit direct forms *fail*. This behavior can be traced to the pole sensitivity formula Eq. (P7.12) which can cause large shifts in the poles even though $\mathbf{da}$ is small.

Table P7.2 shows the poles of the exact and approximate filters, that is, the roots of the denominator polynomials $\mathbf{a}$, 5- and 4-digit $\hat{\mathbf{a}}$, and of the rounded SOS polynomials $\hat{\mathbf{a}}_0$, $\hat{\mathbf{a}}_1$, and $\hat{\mathbf{a}}_2$.

For the 5-digit direct form, the filter remains stable. But, for the 4-digit direct form, one root of $\hat{\mathbf{a}}$ has moved outside the unit circle and the third conjugate pair of roots has been replaced by two real-valued ones, one almost *on* the unit circle. Thus, the quantized coefficients $\hat{\mathbf{a}}$ correspond to an unstable filter. By contrast, the roots of the 4-digit quantized cascaded form $\hat{\mathbf{a}}_{\text{cas}}$ remain inside the unit circle, and in fact they are practically equal to the unquantized ones. Figure P7.27 shows the impulse responses of the exact filter based on $\mathbf{a}$ and the 4-digit-precision filter based on $\hat{\mathbf{a}}$, which is unstable.
The impulse responses of the 5-digit case and the 4-digit cascade case based on $\hat{\mathbf{a}}_{\text{cas}}$ are shown in Fig. P7.28. The magnitude responses of the full precision, 5-digit direct, and 4-digit cascade cases are shown in Fig. P7.29. Note that the 5-digit direct no longer meets the passband specs; it has a passband ripple of almost 2 dB instead of the required 1 dB. The 4-digit cascade response, on the other hand, meets the specifications.

full precision	magnitude
$0.933557 \pm j0.301259$	0.980962
$0.923398 \pm j0.215739$	0.948265
$0.925911 \pm j0.078311$	0.929217

5-digit direct	magnitude
$0.931610 \pm j0.300414$	0.978849
$0.927787 \pm j0.216576$	0.952730
$0.923469 \pm j0.079185$	0.926857

4-digit direct	magnitude
$0.967208 \pm j0.284105$	1.008071
$0.888774 \pm j0.266869$	0.927976
0.999999	0.999999
0.853736	0.853736

4-digit cascade	magnitude
$0.933552 \pm j0.301303$	0.980971
$0.923396 \pm j0.215719$	0.948258
$0.925902 \pm j0.078152$	0.929194

Table P7.2 Filter poles.

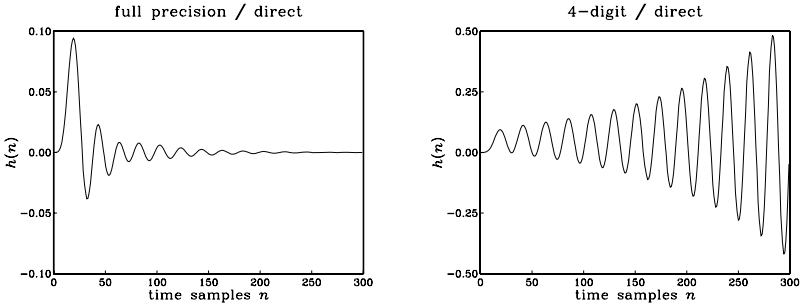


Fig. P7.27 Impulse responses of full-precision and 4-digit-precision filters. Problem 7.21.

Problem 7.22

See the solution of Problem 7.20.

Problem 7.23

See the solution of Problem 7.20.

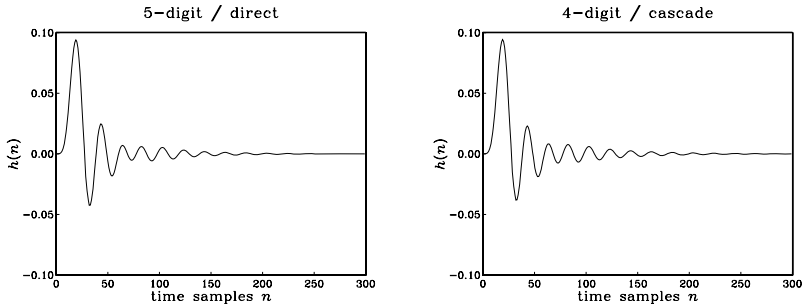


Fig. P7.28 Impulse responses of 5-digit direct and 4-digit cascade filters. Problem 7.21.

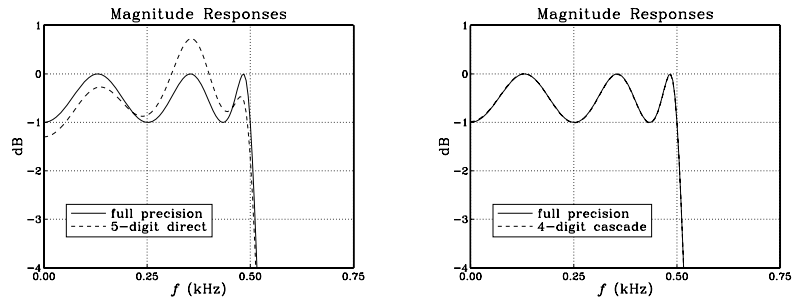


Fig. P7.29 Magnitude responses of exact, 5-digit direct, and 4-digit cascade. Problem 7.21.

Problem 7.24

The transfer function is:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}}{1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3}} \tag{P7.15}$$

Transposed realizations of FIR filters are obtained by the four transposition rules of exchanging input with output, reversing all flows, replacing nodes by adders, and adders by nodes. These rules can be applied to the canonical or direct realizations of IIR filters.

The I/O difference equations describing the time-domain operation of the realization and the corresponding sample processing algorithm can be obtained by introducing as internal states the contents of the three delay registers at time n: $v_1(n)$, $v_2(n)$, $v_3(n)$. For indexing convenience, we also introduce the quantity $v_0(n)$ which is the overall output obtained at the top adder, that is,

$$v_0(n) = y(n) = b_0 x(n) + v_1(n)$$

The output of each delay is the delayed version of the output of the adder below it, for example,

$$v_1(n) = b_1 x(n-1) - a_1 v_0(n-1) + v_2(n-1) = b_1 x(n-1) - a_1 y(n-1) + v_2(n-1)$$

or, advancing it to the next time instant:

$$v_1(n+1) = b_1 x(n) - a_1 v_0(n) + v_2(n) = b_1 x(n) - a_1 y(n) + v_2(n)$$

Therefore, the overall I/O system, which includes the updating of the internal states, will be:

$$\boxed{\begin{aligned} y(n) &= v_0(n) = b_0 x(n) + v_1(n) \\ v_1(n+1) &= b_1 x(n) - a_1 y(n) + v_2(n) \\ v_2(n+1) &= b_2 x(n) - a_2 y(n) + v_3(n) \\ v_3(n+1) &= b_3 x(n) - a_3 y(n) \end{aligned}} \tag{P7.16}$$

which leads to the sample processing algorithm:

for each input sample x do:
$y = v_0 = b_0 x + v_1$
$v_1 = b_1 x - a_1 y + v_2$
$v_2 = b_2 x - a_2 y + v_3$
$v_3 = b_3 x - a_3 y$

Problem 7.25

In the case of a general transfer function of the type of Eq. (7.1.4), we have the sample processing algorithm, where we assumed $L = M$ for simplicity:

for each input sample x do:
$y = v_0 = b_0 x + v_1$
for $i = 1, 2, \ldots, M-1$ do:
$v_i = b_i x - a_i v_0 + v_{i+1}$
$v_M = b_M x - a_M v_0$ (P7.17)

The following C function `transp.c` is an implementation.

```
/* transp.c - IIR filtering in transposed of canonical form */

double transp(M, a, b, v, x)              usage: y = transp(M, a, b, v, x);
double *a, *b, *v, x;                     v = internal state vector
int M;                                    a,b have same order M
{
       int i;

       v[0] = b[0] * x + v[1];                          output y(n) = v0(n)

       for (i=1; i<=M-1; i++)                           state updating
           v[i] = b[i] * x - a[i] * v[0] + v[i+1];

       v[M] = b[M] * x - a[M] * v[0];

       return v[0];
}
```

Its usage is the same as that of `can` or `dir`. The $(M+1)$-dimensional array $\mathbf{v} = [v_0, v_1, \ldots, v_M]$ must be declared and allocated in the main program, for example, by

```
double *v;
v = (double *) calloc(M+1, sizeof(double));
```

Transposed realizations are not intuitively obvious, but their overall implementation in terms of Eq. (P7.17) has a certain simplicity and elegance. It corresponds to writing the transfer function of the filter in a nested form, which is similar to Hörner's rule of evaluating a polynomial. For example, the transfer function (P7.15) can be written as:

$$H(z) = -(a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3})H(z) + b_0 + b_1 z^{-1} + b_2 z^{-2} + b_3 z^{-3}$$

which can be written in the nested form:

$$H = b_0 + z^{-1}\Big[\,(b_1 - a_1 H) + z^{-1}\big[\,(b_2 - a_2 H) + z^{-1}(b_3 - a_3 H)\,\big]\Big]$$

Multiplying both sides by the z-transform $X(z)$ of the input $x(n)$, and using $Y(z) = H(z)X(z)$, we obtain the following nested form for the computation of the output, which is basically the transposed form:

$$Y = b_0 X + z^{-1}\Big[\,(b_1 X - a_1 Y) + z^{-1}\big[\,(b_2 X - a_2 Y) + z^{-1}(b_3 X - a_3 Y)\,\big]\Big]$$

The transposed form can also be used in the cascade realization, where each second order section will be realized in its transposed form. The sample processing algorithm of a 2nd order section is simply:

for each input sample x do:
$y = v_0 = b_0 x + v_1$
$v_1 = b_1 x - a_1 y + v_2$
$v_2 = b_2 x - a_2 y$

which can replace Eq. (7.2.2).

Problem 7.26

The following program implements filtering in the transposed form, calling the routine `transp.c` of Problem 7.25. It replaces `canfilt.c` of Problem 7.14. Because `transp` assumes that **a** and **b** have the same length M, this program aborts if the lengths are unequal.

```
/* trspfilt.c - IIR filtering in transposed form */

#include <stdlib.h>
#include <stdio.h>

#define MAX 64                                  initial allocation size

double transp();

void main(int argc, char **argv)
{
    FILE *fpa, *fpb;                            coefficient files
    double *a, *b, *v, x, y;                    coeffs., state, input, output
    int M, L, i;
    int max = MAX, dmax = MAX;                  initial allocation and increment

    if (argc != 3) {
        fprintf(stderr, "Usage: trspfilt a.dat b.dat <x.dat >y.dat\n");
        fprintf(stderr, "coefficient vectors a,b must have same length");
        exit(0);
        }

    if ((fpa = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "Can't open filter file: %s\n", argv[1]);
        exit(0);
        }

    if ((fpb = fopen(argv[2], "r")) == NULL) {
        fprintf(stderr, "Can't open filter file: %s\n", argv[2]);
        exit(0);
        }

    a = (double *) calloc(max + 1, sizeof(double));
    for (M=0;; M++) {
        if (M == max) {
            max += dmax;
            a = (double *) realloc((char *) a, (max + 1) * sizeof(double));
            }
        if (fscanf(fpa, "%lf", a + M) == EOF) break;
        }
    M--;
    a = (double *) realloc((char *) a, (M + 1) * sizeof(double));

    b = (double *) calloc(max + 1, sizeof(double));
    for (L=0;; L++) {
        if (L == max) {
            max += dmax;
            b = (double *) realloc((char *) b, (max + 1) * sizeof(double));
            }
        if (fscanf(fpb, "%lf", b + L) == EOF) break;
        }
    L--;
    b = (double *) realloc((char *) b, (L + 1) * sizeof(double));

    if (M != L) {
        fprintf(stderr, "coefficient vectors a,b must have same length");
        exit(0);
        }

    v = (double *) calloc(M + 1, sizeof(double));

    while(scanf("%lf", &x) != EOF) {            process input samples
        y = transp(M, a, b, v, x);
        printf("%lf\n", y);
        }
}
```

Problem 7.27

The transfer function is:

$$H(z) = \frac{9 - 4z^{-2} + 4z^{-4}}{1 + 0.84z^{-2} + 0.25z^{-4}} = \frac{3 - 4z^{-1} + 2z^{-2}}{1 - 0.4z^{-1} + 0.5z^{-2}} \cdot \frac{3 + 4z^{-1} + 2z^{-2}}{1 + 0.4z^{-1} + 0.5z^{-2}}$$

The transpose of the canonical form is shown in Fig. P7.30. The I/O difference equations are:

$$
\begin{aligned}
y(n) &= 9x(n) + v_1(n) \\
v_1(n+1) &= v_2(n) \\
v_2(n+1) &= -4x(n) - 0.84y(n) + v_3(n) \\
v_3(n+1) &= v_4(n) \\
v_4(n+1) &= 4x(n) - 0.25y(n)
\end{aligned}
$$

and the corresponding sample processing algorithm:

for each input sample x do:
$y = 9x + v_1$
$v_1 = v_2$
$v_2 = -4x - 0.84y + v_3$
$v_3 = v_4$
$v_4 = 4x - 0.25y$

The cascade form with both 2nd order sections realized in their transposed form is shown in Fig. P7.31. The I/O difference equations are in this case:

$$
\begin{aligned}
x_1(n) &= 3x(n) + v_{01}(n) \\
v_{01}(n+1) &= -4x(n) + 0.4x_1(n) + v_{02}(n) \\
v_{02}(n+1) &= 2x(n) - 0.5x_1(n) \\
y(n) &= 3x_1(n) + v_{11}(n) \\
v_{11}(n+1) &= 4x_1(n) - 0.4y(n) + v_{12}(n) \\
v_{12}(n+1) &= 2x_1(n) - 0.5y(n)
\end{aligned}
$$

The sample processing algorithm will be:

for each input sample x do:
$x_1 = 3x + v_{01}$
$v_{01} = -4x + 0.4x_1 + v_{02}$
$v_{02} = 2x - 0.5x_1$
$y = 3x_1 + v_{11}$
$v_{11} = 4x_1 - 0.4y + v_{12}$
$v_{12} = 2x_1 - 0.5y$

Problem 7.28

The overall, closed-loop, transfer function of the feedback system can be worked out easily using z-transforms. The general approach to analyzing this, and any other, block diagram is to assign names to all signal lines that do not already have names and then set up the various adder and filtering equations.

Consider case A, with signal names assigned as in Fig. P7.32 Then, the adder and filtering equations are in the time and z domains:

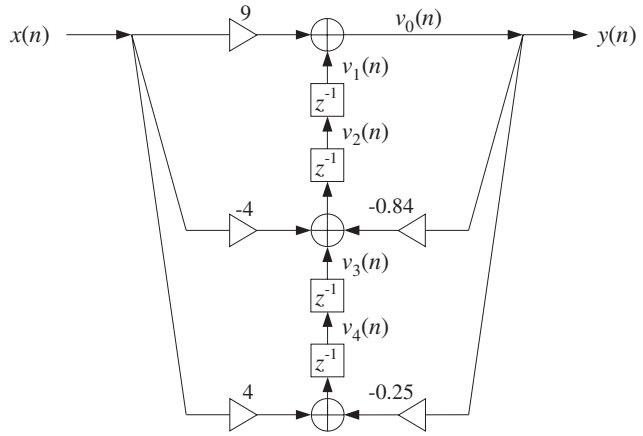


Fig. P7.30 Transposed of canonical form.

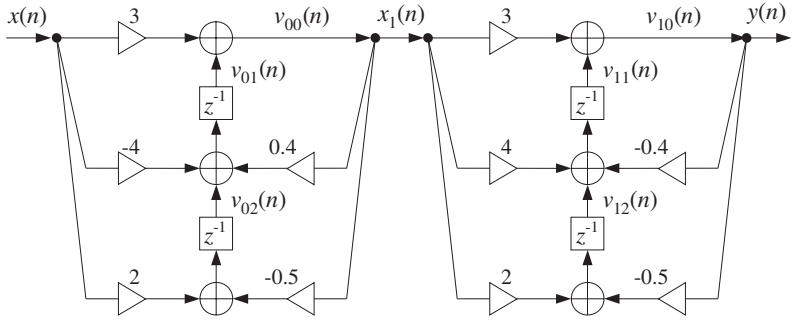


Fig. P7.31 Cascade form with transposed sections.

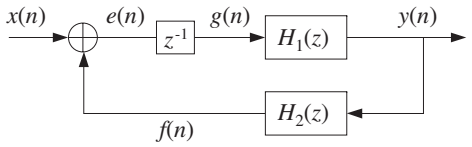


Fig. P7.32 Case A of Problem 7.28

$$
\begin{aligned}
e(n) &= x(n) + f(n) & E(z) &= X(z) + F(z) \\
g(n) &= e(n-1) & G(z) &= z^{-1}E(z) \\
y(n) &= h_1(n) * g(n) \quad \Rightarrow & Y(z) &= H_1(z)G(z) \\
f(n) &= h_2(n) * y(n) & F(z) &= H_2(z)Y(z)
\end{aligned}
$$

Eliminating $F(z)$ and $E(z)$ in favor of $X(z)$ and $Y(z)$ gives:

$$Y(z) = z^{-1}H_1(z)E(z) = z^{-1}H_1(z)\big(X(z)+F(z)\big) = z^{-1}H_1(z)\big(X(z)+H_2(z)Y(z)\big)$$

and

$$\big(1 - z^{-1}H_1(z)H_2(z)\big)Y(z) = z^{-1}H_1(z)X(z)$$

Solving for the closed-loop transfer function $H(z) = Y(z)/X(z)$, we find:

$$\boxed{H(z) = \frac{z^{-1}H_1(z)}{1 - z^{-1}H_1(z)H_2(z)}}$$

Case B has the same transfer function $H(z)$. Cases C and D have:

$$\boxed{H(z) = \frac{H_1(z)}{1 - z^{-1}H_1(z)H_2(z)}}$$

Problem 7.29

The overall transfer function of this system can be computed using the results of Problem 7.28 and the individual transfer functions:

$$H_1(z) = \frac{1+z^{-1}}{1-0.5z^{-1}}, \qquad H_2(z) = z^{-1}\frac{0.4(1-z^{-1})}{1-0.4z^{-1}}$$

$$H(z) = \frac{H_1(z)}{1-H_1(z)H_2(z)} = \frac{\dfrac{1+z^{-1}}{1-0.5z^{-1}}}{1 - \dfrac{1+z^{-1}}{1-0.5z^{-1}} \cdot \dfrac{0.4(1-z^{-1})z^{-1}}{1-0.4z^{-1}}}$$

which gives

$$H(z) = \frac{1+0.6z^{-1}-0.4z^{-2}}{1-1.3z^{-1}+0.2z^{-2}+0.4z^{-3}}$$

In case A, we assign internal state variables as shown in Fig. P7.33.
The difference equations and sample processing algorithm, written in the right computational order, are:

$$\begin{aligned} e(n) &= x(n)+f(n-1) \\ w(n) &= 0.5w(n-1)+e(n) \\ y(n) &= w(n)+w(n-1) \\ v(n) &= 0.4v(n-1)+y(n) \\ f(n) &= 0.4v(n)-0.4v(n-1) \end{aligned}$$

for each input sample x do:
- $e = x + f_1$
- $w_0 = 0.5w_1 + e$
- $y = w_0 + w_1$
- $v_0 = 0.4v_1 + y$
- $f_0 = 0.4(v_0 - v_1)$
- $f_1 = f_0$ (update delays)
- $w_1 = w_0$
- $v_1 = v_0$

Notice how the extra delay factor z^{-1} in $H_2(z)$ delays the output $f(n)$ and allows the start of the computational cycle with $e(n) = x(n)+f(n-1)$. The output $f(n)$ is computed last. If this delay were not there, we would have $e(n) = x(n)+f(n)$, which is not possible because $f(n)$ is not yet

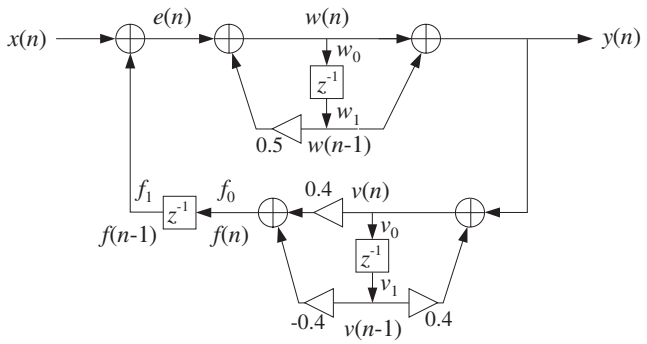


Fig. P7.33 Case A of Problem 7.29.

computed.

In case B, shown in Fig. P7.34, the overall transfer function remains the same. The change causes a re-ordering of the computational cycle. The difference equations and corresponding sample processing algorithm are now:

$$\begin{aligned} v(n) &= 0.4v(n-1)+y(n-1) \\ f(n) &= 0.4v(n)-0.4v(n-1) \\ e(n) &= x(n)+f(n) \\ w(n) &= 0.5w(n-1)+e(n) \\ y(n) &= w(n)+w(n-1) \end{aligned}$$

for each input sample x do:
- $v_0 = 0.4v_1 + y_1$
- $f = 0.4(v_0 - v_1)$
- $e = x + f$
- $w_0 = 0.5w_1 + e$
- $y = w_0 + w_1$
- $y_1 = y$ (update delays)
- $w_1 = w_0$
- $v_1 = v_0$

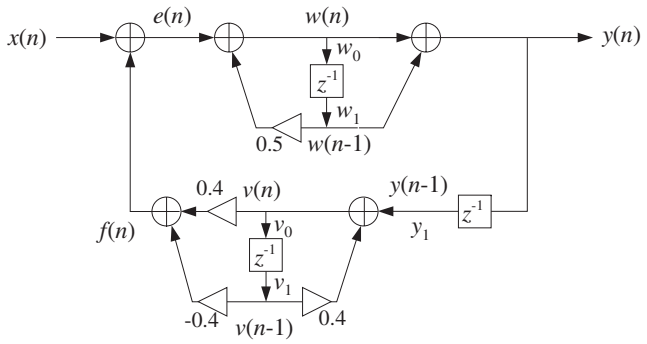


Fig. P7.34 Case B of Problem 7.29.

Now, the start of the computational cycle is $v(n) = 0.4v(n-1)+y(n-1)$, where both quantities

$v(n-1)$ and $y(n-1)$ are available from the previous time step. Note that $y(n)$ is computed last and then put into the feedback delay to be used in the next time step.

Problem 7.30

The filter $G(z)$ can be implemented by the input/output sample processing algorithm of its canonical form:

```
for each x do:
    y = 0.2x + 0.8w1
    w1 = y
```

where w_1 is its internal state, that is, $w_1(n) = y(n-1)$ in the difference equation: $y(n) = 0.2x(n) + 0.8y(n-1)$. Fig. P7.35 shows the three cases with explicitly assigned signal labels. Identifying the proper input and output to the filter $G(z)$ and using the above sample processing algorithm, we obtain the implementation of the three cases:

```
for each x do:
    u = 0.2v4 + 0.8w1
    w1 = u
    y = v0 = x + u
    delay(4, v)
```

```
for each x do:
    y = x + v4
    v0 = 0.2y + 0.8w1
    w1 = v0
    delay(4, v)
```

```
for each x do:
    y = u0 = x + v2
    v0 = 0.2u2 + 0.8w1
    w1 = v0
    delay(2, v), delay(2, u)
```

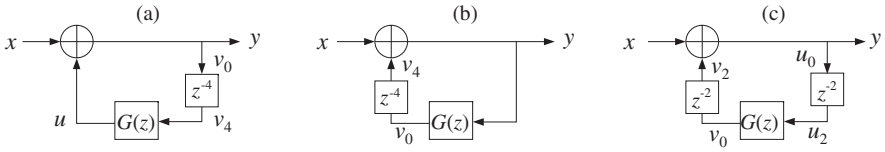


Fig. P7.35 Feedback systems of Problem 7.30.

Chapter 8 Problems

Problem 8.1

The transfer function is obtained by the period-3 repetition of the sequence $b(n) = [0, 1, 2]$, that is, in the time- and z-domains:

$$h(n) = b(n) + b(n-3) + b(n-6) + \cdots$$

$$H(z) = B(z) + z^{-3}B(z) + z^{-6}B(z) + \cdots = \frac{B(z)}{1-z^{-3}} = \frac{z^{-1}+2z^{-2}}{1-z^{-3}}$$

The direct and canonical realizations are shown in Fig. P8.1. Their sample processing algorithms are:

```
for each input x do:
    v0 = x
    y = w0 = v1 + 2v2 + w3
    delay(2, v)
    delay(3, w)
```

```
for each input x do:
    w0 = x + w3
    y = w1 + 2w2
    delay(3, w)
```

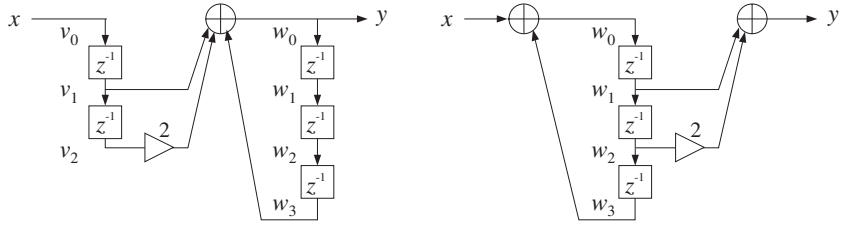


Fig. P8.1 Direct and canonical realizations of Problem 8.1.

For the direct form realization, if the input is a unit impulse so that $x = 1$ at the first iteration of the algorithm and zero for all other iterations, then the first three iterations will fill the feedback delay buffer **w** with the desired waveform samples. From then on, the feed-forward part of the algorithm will be zero, giving rise to the following generation algorithm, expressed in its of the linear and circular buffer versions:

```
repeat forever:
    w0 = w3
    delay(3, w)
```

```
repeat forever:
    *p = tap(3, w, p, 3)
    cdelay(3, w, &p)
```

The following table shows the initialization and steady part of the linear buffer case. The columns v_0, v_1, v_2 are the input impulse and its delays: $\delta(n)$, $\delta(n-1)$, $\delta(n-2)$. At each iteration, w_0 is replaced by the value of w_3 and only then is the row of w's shifted to the right to give the values of w_1, w_2, w_3 of the next row. The w-columns are the successively delayed output signals $y(n)$, $y(n-1)$, $y(n-2)$, $y(n-3)$:

n	v_0	v_1	v_2	w_0	w_1	w_2	w_3	$y = w_0$
0	1	0	0	0	0	0	0	0
1	0	1	0	1	0	0	0	1
2	0	0	1	2	1	0	0	2
3	0	0	0	0	2	1	0	0
4	0	0	0	1	0	2	1	1
5	0	0	0	2	1	0	2	2
6	0	0	0	0	2	1	0	0
7	0	0	0	1	0	2	1	1
8	0	0	0	2	1	0	2	2

The circular version is given in the following table. The buffer entry that contains the current output sample is shown with an up-arrow symbol and it corresponds to the entry w_q defined by the circulating value of q:

n	q	w_0	w_1	w_2	w_3	y
0	0	↑0	0	0	0	0
1	3	0	0	0	↑1	1
2	2	0	0	↑2	1	2
3	1	0	↑0	2	1	0

n	q	w_0	w_1	w_2	w_3	y
4	0	↑1	0	2	1	1
5	3	1	0	2	↑2	2
6	2	1	0	↑0	2	0
7	1	1	↑1	0	2	1

n	q	w_0	w_1	w_2	w_3	y
8	0	↑2	1	0	2	2
9	3	2	1	0	↑0	0
10	2	2	1	↑1	0	1
11	1	2	↑2	1	0	2

n	q	w_0	w_1	w_2	w_3	y
12	0	↑0	2	1	0	0
13	3	0	2	1	↑1	1
14	2	0	2	↑2	1	2
15	1	0	↑0	2	1	0

The table entries circulate every $D(D+1) = 12$ iterations. Thus, entry $n = 3$ and $n = 15$ will be the same. We used 15 iterations because they contain one complete cycle of the table entries after the first 3 initializing iterations.

Problem 8.2

Expanding the denominator in powers of z^{-5}, will cause the period-5 replication of the numerator sequence $\mathbf{b} = [1, 2, 3, -4, -5]$. The direct and canonical realizations are shown in Fig. P8.2. Their sample processing algorithms are:

for each input x do:
$v_0 = x$
$w_0 = v_0 + 2v_1 + 3v_2 - 4v_3 - 5v_4 + w_5$
$y = w_0$
delay$(4, \mathbf{v})$
delay$(5, \mathbf{w})$

for each input x do:
$w_0 = x + w_5$
$y = w_0 + 2w_1 + 3w_2 - 4w_3 - 5w_4$
delay$(5, \mathbf{w})$

The steady parts of the linear and circular buffer generation algorithms are:

repeat forever:
$w_0 = w_5$
delay$(5, \mathbf{w})$

repeat forever:
$*p =$ tap$(5, \mathbf{w}, p, 5)$
cdelay$(5, \mathbf{w}, \&p)$

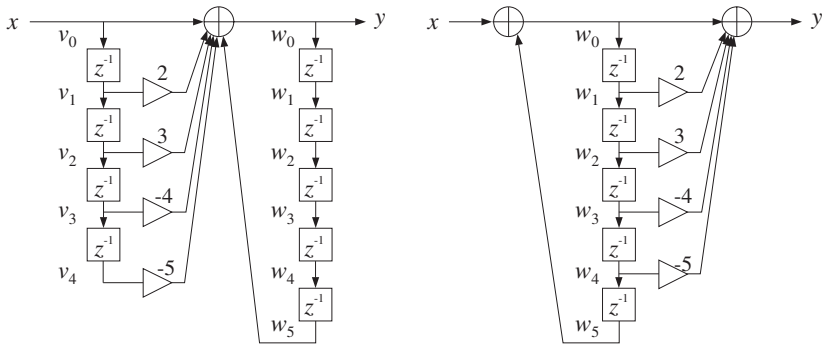


Fig. P8.2 Direct and canonical realizations of Problem 8.2.

The following table shows the initialization and steady parts of the linear buffer case:

n	v_0	v_1	v_2	v_3	v_4	w_0	w_1	w_2	w_3	w_4	w_5	y
0	1	0	0	0	0	1	0	0	0	0	0	1
1	0	1	0	0	0	2	1	0	0	0	0	2
2	0	0	1	0	0	3	2	1	0	0	0	3
3	0	0	0	1	0	−4	3	2	1	0	0	−4
4	0	0	0	0	1	−5	−4	3	2	1	0	−5
5	0	0	0	0	0	1	−5	−4	3	2	1	1
6	0	0	0	0	0	2	1	−5	−4	3	2	2
7	0	0	0	0	0	3	2	1	−5	−4	3	3
8	0	0	0	0	0	−4	3	2	1	−5	−4	−4
9	0	0	0	0	0	−5	−4	3	2	1	−5	−5
10	0	0	0	0	0	1	−5	−4	3	2	1	1
11	0	0	0	0	0	2	1	−5	−4	3	2	2
12	0	0	0	0	0	3	2	1	−5	−4	3	3
13	0	0	0	0	0	−4	3	2	1	−5	−4	−4
14	0	0	0	0	0	−5	−4	3	2	1	−5	−5

The circular version is given in the following table. The buffer entry that contains the current output sample is shown with an up-arrow symbol and it corresponds to the entry w_q defined by the circulating value of q:

n	q	w_0	w_1	w_2	w_3	w_4	w_5	y
0	0	1↑	0	0	0	0	0	1
1	5	1	0	0	0	0	2↑	2
2	4	1	0	0	0	3↑	2	3
3	3	1	0	0	−4↑	3	2	−4
4	2	1	0	−5↑	−4	3	2	−5
5	1	1	1↑	−5	−4	3	2	1

n	q	w_0	w_1	w_2	w_3	w_4	w_5	y
6	0	$2\uparrow$	1	-5	-4	3	2	2
7	5	2	1	-5	-4	3	$3\uparrow$	3
7	4	2	1	-5	-4	$-4\uparrow$	3	-4
9	3	2	1	-5	$-5\uparrow$	-4	3	-5
10	2	2	1	$1\uparrow$	-5	-4	3	1
11	1	2	$2\uparrow$	1	-5	-4	3	2

n	q	w_0	w_1	w_2	w_3	w_4	w_5	y
12	0	$3\uparrow$	2	1	-5	-4	3	3
13	5	3	2	1	-5	-4	$-4\uparrow$	-4
14	4	3	2	1	-5	$-5\uparrow$	-4	-5
15	3	3	2	1	$1\uparrow$	-5	-4	1
16	2	3	2	$2\uparrow$	1	-5	-4	2
17	1	3	$3\uparrow$	2	1	-5	-4	3

n	q	w_0	w_1	w_2	w_3	w_4	w_5	y
18	0	$-4\uparrow$	3	2	1	-5	-4	-4
19	5	-4	3	2	1	-5	$-5\uparrow$	-5
20	4	-4	3	2	1	$1\uparrow$	-5	1
21	3	-4	3	2	$2\uparrow$	1	-5	2
22	2	-4	3	$3\uparrow$	2	1	-5	3
23	1	-4	$-4\uparrow$	3	2	1	-5	-4

n	q	w_0	w_1	w_2	w_3	w_4	w_5	y
24	0	$-5\uparrow$	-4	3	2	1	-5	-5
25	5	-5	-4	3	2	1	$1\uparrow$	1
26	4	-5	-4	3	2	$2\uparrow$	1	2
27	3	-5	-4	3	$3\uparrow$	2	1	3
28	2	-5	-4	$-4\uparrow$	3	2	1	-4
29	1	-5	$-5\uparrow$	-4	3	2	1	-5

n	q	w_0	w_1	w_2	w_3	w_4	w_5	y
30	0	$1\uparrow$	-5	-4	3	2	1	1
31	5	1	-5	-4	3	2	$2\uparrow$	2
32	4	1	-5	-4	3	$3\uparrow$	2	3
33	3	1	-5	-4	$-4\uparrow$	3	2	-4
34	2	1	-5	$-5\uparrow$	-4	3	2	-5
35	1	1	$1\uparrow$	-5	-4	3	2	1

The table entries circulate every $D(D+1)=30$ iterations. Thus, entry $n=5$ and $n=35$ will be the same. We used 35 iterations because they contain one complete cycle of the table entries after the first 5 initializing iterations.

Problem 8.3

The shift is $c = D/d = 8/5 = 1.6$. The $d=5$ possible values of the offset index q are obtained by iterating Eq. (8.1.36):

$$q_0 = 0$$

$$q_1 = q_0 - c = -\frac{8}{5} \equiv 8 - \frac{8}{5} = \frac{32}{5} = 6\frac{2}{5}$$

$$q_2 = q_1 - c = \frac{32}{5} - \frac{8}{5} = \frac{24}{5} = 4\frac{4}{5}$$

$$q_3 = q_2 - c = \frac{24}{5} - \frac{8}{5} = \frac{16}{5} = 3\frac{1}{5}$$

$$q_4 = q_3 - c = \frac{16}{5} - \frac{8}{5} = \frac{8}{5} = 1\frac{3}{5}$$

Fig. P8.3 shows the relative locations of the q's with respect to the circular buffer. The five q-arrows are now at relative angles $\omega = 2\pi/5$.

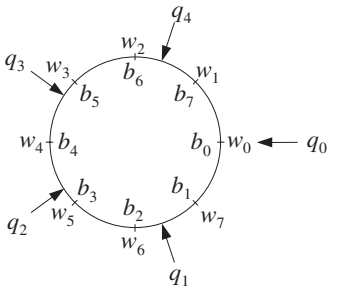


Fig. P8.3 Successive positions of q when $d=5$.

Down-truncation gives the following integer values for the q's and corresponding buffer entries:

$$[q_0, q_1, q_2, q_3, q_4] = [0, 6, 4, 3, 1]$$

$$[w[0], w[6], w[4], w[3], w[1]] = [b_0, b_2, b_4, b_5, b_7]$$

Up-truncation gives:

$$[q_0, q_1, q_2, q_3, q_4] = [0, 7, 5, 4, 2]$$

$$[w[0], w[7], w[5], w[4], w[2]] = [b_0, b_1, b_3, b_4, b_6]$$

Rounding to the nearest integer gives:

$$[q_0, q_1, q_2, q_3, q_4] = [0, 6, 5, 3, 2]$$

$$[w[0], w[6], w[5], w[3], w[2]] = [b_0, b_2, b_3, b_5, b_6]$$

For the linear interpolation case, we have the outputs:

$$
\begin{aligned}
q_0 = 0 \quad &\Rightarrow \quad y_0 = w[0] = b_0 \\
6 < q_1 < 7 \quad &\Rightarrow \quad y_1 = w[6] + (q_1 - 6)\,(w[7] - w[6]) = \frac{2}{5}b_1 + \frac{3}{5}b_2 \\
4 < q_2 < 5 \quad &\Rightarrow \quad y_2 = w[4] + (q_2 - 4)\,(w[5] - w[4]) = \frac{4}{5}b_3 + \frac{1}{5}b_4 \\
3 < q_3 < 4 \quad &\Rightarrow \quad y_3 = w[3] + (q_3 - 3)\,(w[4] - w[3]) = \frac{1}{5}b_4 + \frac{4}{5}b_5 \\
1 < q_4 < 2 \quad &\Rightarrow \quad y_4 = w[1] + (q_4 - 1)\,(w[2] - w[1]) = \frac{3}{5}b_6 + \frac{2}{5}b_7
\end{aligned}
$$

Thus, depending on the output method, the following period-5 subsequences will be produced:

$$
\begin{array}{ll}
[b_0, b_2, b_4, b_5, b_7, b_0, b_2, b_4, b_5, b_7, \dots\,] & \text{(truncate down)} \\
[b_0, b_1, b_3, b_4, b_6, b_0, b_1, b_3, b_4, b_6, \dots\,] & \text{(truncate up)} \\
[b_0, b_2, b_3, b_5, b_6, b_0, b_2, b_3, b_5, b_6, \dots\,] & \text{(round)} \\
[y_0, y_1, y_2, y_3, y_4, y_0, y_1, y_2, y_3, y_4, \dots\,] & \text{(interpolation)}
\end{array}
$$

where the y's were given above.

Problem 8.4

The shift is $c = D/d = 8/6 = 4/3$. The $d = 6$ possible values of the offset index q are obtained by iterating Eq. (8.1.36):

$$
\begin{aligned}
q_0 &= 0 \\
q_1 &= q_0 - c = -\frac{4}{3} \equiv 8 - \frac{4}{3} = \frac{20}{3} = 6\frac{2}{3} \\
q_2 &= q_1 - c = \frac{20}{3} - \frac{4}{3} = \frac{16}{3} = 5\frac{1}{3} \\
q_3 &= q_2 - c = \frac{16}{3} - \frac{4}{3} = \frac{12}{3} = 4 \\
q_4 &= q_3 - c = \frac{12}{3} - \frac{4}{3} = \frac{8}{3} = 2\frac{2}{3} \\
q_5 &= q_4 - c = \frac{8}{3} - \frac{4}{3} = \frac{4}{3} = 1\frac{1}{3}
\end{aligned}
$$

Fig. P8.4 shows the relative locations of the q's with respect to the circular buffer. The six q-arrows are now at relative angles $\omega = 2\pi/6$.
Down-truncation gives the following integer values for the q's and corresponding buffer entries:

$$[q_0, q_1, q_2, q_3, q_4, q_5] = [0, 6, 5, 4, 2, 1]$$

$$[w[0], w[6], w[5], w[4], w[2], w[1]] = [b_0, b_2, b_3, b_4, b_6, b_7]$$

Up-truncation gives:

$$[q_0, q_1, q_2, q_3, q_4, q_5] = [0, 7, 6, 4, 3, 2]$$

$$[w[0], w[7], w[6], w[4], w[3], w[2]] = [b_0, b_1, b_2, b_4, b_5, b_6]$$

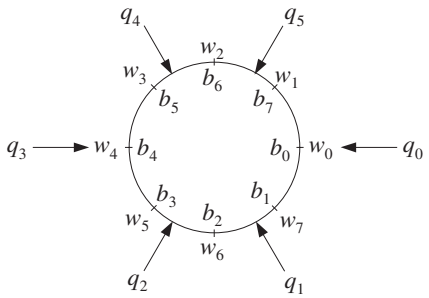


Fig. P8.4 Successive positions of q when $d = 6$.

Rounding to the nearest integer gives:

$$[q_0, q_1, q_2, q_3, q_4, q_5] = [0, 7, 5, 4, 3, 1]$$

$$[w[0], w[7], w[5], w[4], w[3], w[1]] = [b_0, b_1, b_3, b_4, b_5, b_7]$$

For the linear interpolation case, we have the outputs:

$$
\begin{aligned}
q_0 = 0 \quad &\Rightarrow \quad y_0 = w[0] = b_0 \\
6 < q_1 < 7 \quad &\Rightarrow \quad y_1 = w[6] + (q_1 - 6)\,(w[7] - w[6]) = \frac{2}{3}b_1 + \frac{1}{3}b_2 \\
5 < q_2 < 6 \quad &\Rightarrow \quad y_2 = w[5] + (q_2 - 5)\,(w[6] - w[5]) = \frac{1}{3}b_2 + \frac{2}{3}b_3 \\
q_3 = 4 \quad &\Rightarrow \quad y_3 = w[4] = b_4 \\
2 < q_4 < 3 \quad &\Rightarrow \quad y_4 = w[2] + (q_4 - 2)\,(w[3] - w[2]) = \frac{2}{3}b_5 + \frac{1}{3}b_6 \\
1 < q_5 < 2 \quad &\Rightarrow \quad y_5 = w[1] + (q_5 - 1)\,(w[2] - w[1]) = \frac{1}{3}b_6 + \frac{2}{3}b_7
\end{aligned}
$$

Thus, depending on the output method, the following period-5 subsequences will be produced:

$$
\begin{array}{ll}
[b_0, b_2, b_3, b_4, b_6, b_7, b_0, b_2, b_3, b_4, b_6, b_7, \dots\,] & \text{(truncate down)} \\
[b_0, b_1, b_2, b_4, b_5, b_6, b_0, b_1, b_2, b_4, b_5, b_6, \dots\,] & \text{(truncate up)} \\
[b_0, b_1, b_3, b_4, b_5, b_7, b_0, b_1, b_3, b_4, b_5, b_7, \dots\,] & \text{(round)} \\
[y_0, y_1, y_2, y_3, y_4, y_5, y_0, y_1, y_2, y_3, y_4, y_5, \dots\,] & \text{(interpolate)}
\end{array}
$$

where the y's are given above.

Problem 8.5

This problem is similar to Example 8.1.2. Referring to Fig. 8.1.14, we have for the cases of truncating down, up, and rounding:

$$
\begin{array}{ll}
[b_0, b_3, b_6, b_0, b_3, b_6, \dots\,] = [1, 4, 7, 1, 4, 7, \dots\,] & \text{(truncate down)} \\
[b_0, b_2, b_5, b_0, b_2, b_5, \dots\,] = [1, 3, 6, 1, 3, 6, \dots\,] & \text{(truncate up)} \\
[b_0, b_3, b_5, b_0, b_3, b_5, \dots\,] = [1, 4, 6, 1, 4, 6, \dots\,] & \text{(round)}
\end{array}
$$

If we interpolate linearly, we get the sequence:

$$y_0 = b_0 = 1$$

$$y_1 = \frac{1}{3}b_2 + \frac{2}{3}b_3 = \frac{1}{3}3 + \frac{2}{3}4 = \frac{11}{3} = 3.667$$

$$y_2 = \frac{1}{3}b_5 + \frac{2}{3}b_6 = \frac{1}{3}6 + \frac{2}{3}7 = \frac{20}{3} = 6.667$$

so that the period-3 sequence is:

$$[1,\ 3.667,\ 6.667,\ 1,\ 3.667,\ 6.667,\ \dots\ ]$$

Problem 8.6

This problem is similar to Problem 8.3. Inserting the numerical values for the wavetable contents:

$$\mathbf{b} = [b_0, b_1, b_2, b_4, b_5, b_6, b_7] = [1, 2, 3, 4, 5, 6, 7, 8]$$

we find for the period-5 sequences:

$$\begin{array}{ll}
[b_0, b_2, b_4, b_5, b_7, \dots\ ] = [1, 3, 4, 5, 7 \dots\ ] & \text{(truncate down)} \\
[b_0, b_1, b_3, b_4, b_6, \dots\ ] = [1, 2, 3, 5, 7, \dots\ ] & \text{(truncate up)} \\
[b_0, b_2, b_3, b_5, b_6, \dots\ ] = [1, 3, 4, 6, 7, \dots\ ] & \text{(round)} \\
[y_0, y_1, y_2, y_3, y_4, \dots\ ] = [1, 2.6, 4.2, 5.8, 7.4, \dots\ ] & \text{(interpolation)}
\end{array}$$

where

$$y_0 = b_0 = 1$$

$$y_1 = \frac{2}{5}b_1 + \frac{3}{5}b_2 = \frac{2}{5}2 + \frac{3}{5}3 = 2.6$$

$$y_2 = \frac{4}{5}b_3 + \frac{1}{5}b_4 = \frac{4}{5}4 + \frac{1}{5}5 = 4.2$$

$$y_3 = \frac{1}{5}b_4 + \frac{4}{5}b_5 = \frac{1}{5}5 + \frac{1}{5}6 = 5.8$$

$$y_4 = \frac{3}{5}b_6 + \frac{2}{5}b_7 = \frac{3}{5}7 + \frac{2}{5}8 = 7.4$$

The case $d = 6$ is similar to Problem 8.4.

Problem 8.7

An example program that exercises the wavetable routines is listed below. It generates three wavetables: sinusoidal, square wave, and trapezoidal.

```
/* wavgenex.c - circular wavetable example */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

void gdelay2();
double wavgen();
double sine(), square(), trapez();
```



```
void main(int argc, char **argv)
{
   int i, n, D;
   double *w1, q1, *w2, q2, *w3, q3, c, A, F;
   FILE *fp1, *fp2, *fp3;

   if (argc != 4) {
      fprintf(stderr, "usage: wavgenex A, c, D\n");
      fprintf(stderr, "A = wavetable amplitude\n");
      fprintf(stderr, "c = no. of periods in D samples, F=c/D\n");
      fprintf(stderr, "D = wavetable size\n");
      exit(0);
      }

   A = atof(argv[1]);
   c = atof(argv[2]);
   D = atoi(argv[3]);

   F = c / D;

   fp1 = fopen("y1.dat", "w");
   fp2 = fopen("y2.dat", "w");
   fp3 = fopen("y3.dat", "w");

   w1 = (double *) calloc(D, sizeof(double));
   w2 = (double *) calloc(D, sizeof(double));
   w3 = (double *) calloc(D, sizeof(double));

   q1 = 0;
   q2 = 0;
   q3 = 0;

   for (i=0; i<D; i++) {                        /* load wavetables */
      w1[q1] = sine(D, i);                      /* might need the cast w1[(int)q1] */
      w2[q2] = square(D/2, i);
      w3[q3] = trapez(D, 3*D/4, 0, i);
      gdelay2(D-1, 1.0, &q1);
      gdelay2(D-1, 1.0, &q2);
      gdelay2(D-1, 1.0, &q3);
      }

   for (n=0; n<D; n++) {
      fprintf(fp1, "%lf\n", wavgen(D, w1, A, F, &q1));
      fprintf(fp2, "%lf\n", wavgen(D, w2, A, F, &q2));
      fprintf(fp3, "%lf\n", wavgen(D, w3, A, F, &q3));
      }
}
```

Problem 8.8

Partial C code for this problem was given in the text, just before Figs. 8.1.20 and 8.1.21.

Problem 8.9

The zero patterns and magnitude responses $|H(\omega)|$ of the four filters are shown in Fig. 8.9. The first filter has $H(z) = 1 + z^{-8}$. Therefore, its zeros are the solutions of:

$$z^8 = -1 = e^{j\pi}e^{2\pi jk} \qquad \Rightarrow \qquad z_k = e^{j\pi/8}e^{j2\pi k/8}, \qquad k = 0, 1, \ldots, 7$$

That is, the 8th roots of unity rotated by $\pi/8 \equiv= 22.5^o$. The second filter has $H(z) = 1 - z^{-8}$ and thus, its zeros are the 8th roots of unity.

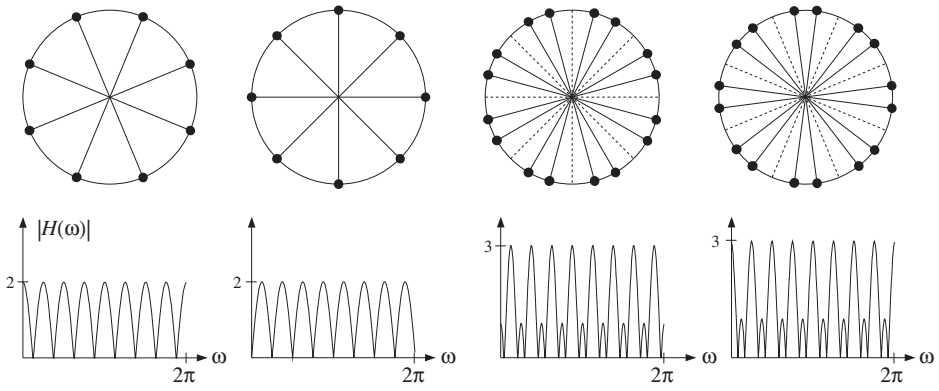


Fig. P8.5 Comb filters of Problem 8.9.

The third and fourth filters can be written as follows, using the finite geometric series:

$$H(z) = 1 - z^{-8} + z^{-16} = \frac{1 + z^{-24}}{1 + z^{-8}}$$

$$H(z) = 1 + z^{-8} + z^{-16} = \frac{1 - z^{-24}}{1 - z^{-8}}$$

The third has roots the 24th roots of unity shifted by $\pi/24 = 7.5^o$, but among them the roots of the denominator $1 + z^{-8}$ must be excluded. This results in 16 zeros. Similarly, the last filter's zeros are the 24th roots of unity with the 8th roots of unity excluded.

Problem 8.10

Typical C code for generating Fig. 8.2.9 and 8.2.11 is given in the text. For the double chorus case, we have the code segment:

```
for (n=0; n<Ntot; n++) {
    d1  =   D * (0.5 + ranl(Dran, u1, &q1, &iseed1));          /* u1 is 2-dim */
    d2  =   D * (0.5 + ranl(Dran, u2, &q2, &iseed2));          /* mean = D/2 */
    a1  = 1 + ranl(Dran, u3, &q3, &iseed3);                    /* mean = 1 */
    a2  = 1 + ranl(Dran, u4, &q4, &iseed4);
    x   = cos(2 * pi * F * n);                                 /* input x(n) */
    xd1 = tapi(D, w, p, d1);                                   /* delay x(n-d1) */
    xd2 = tapi(D, w, p, d2);
    y  = (x + a1 * xd1 + a2 * xd2) / 3;                        /* output */
    *p = x;
    cdelay(D, w, &p);                                          /* update delay */

    fprintf(fpx, "%lf\n", x);                                  /* declaration of fpx */
```



```
    fprintf(fpy, "%lf\n", y);                                  /*   is not shown     */
    fprintf(fpd, "%lf\n", 2*d1/D);
    fprintf(fpa, "%lf\n", a1);
    }
```

Typical output is shown in Fig. P8.6. For the recursive flanger, we have the code fragment:

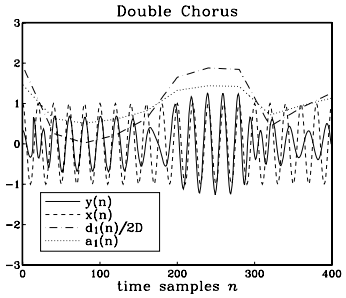


Fig. P8.6 Double chorusing of sinusoidal signal.

```
for (n=0; n<Ntot; n++) {
    d  = 0.50 * D * (1 - cos(2 * pi * Fd * n));
    x  = cos(2 * pi * F * n);
    yd = tapi(D, w, p, d);
    y  = a * yd + x;                                  /* yd is y(n-d) */
    *p = y;
    cdelay(D, w, &p);

    fprintf(fpx, "%lf\n", x);
    fprintf(fpy, "%lf\n", y);
    fprintf(fpd, "%lf\n", d/D);
    }
```

Typical output for $D = 100$ and $D = 10$ is shown in Fig. P8.7.

Problem 8.11

A typical program is as follows:

```
/* reverb.c - plain, allpass, and lowpass reverb */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double plain(), allpass(), lowpass();

void main(int argc, char **argv)
{
    double *w1, *p1;
    double *w2, *p2;
```

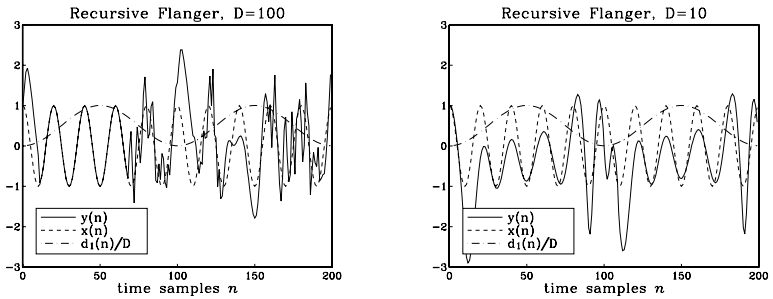


Fig. P8.7 Recursive flanger.

```
double *w3, *p3;

double v[2] = {0., 0.};
double a[2] = {1, -0.5};
double b[2] = {0.3, 0.15};

double a0 = 0.75;
double x, y1, y2, y3, sD;
int D, Ntot, M=1, n;

FILE *fp1, *fp2, *fp3;

fp1 = fopen("y1.dat", "w");
fp2 = fopen("y2.dat", "w");
fp3 = fopen("y3.dat", "w");

if (argc != 3) {
    fprintf(stderr, "usage: reverb D Ntot");      use D=20, Ntot=100
    exit(0);
    }

D    = atoi(argv[1]);
Ntot = atoi(argv[2]);

w1 = (double *) calloc(D+1, sizeof(double));
p1 = w1;
w2 = (double *) calloc(D+1, sizeof(double));
p2 = w2;
w3 = (double *) calloc(D+1, sizeof(double));
p3 = w3;

for (n=0; n<Ntot; n++) {
                                    triangular pulse
    if (n<=5)
        x=n;
    else if (n<=10)
        x = 10-n;
    else
        x=0;
```



```
        y1 = plain(D, w1, &p1, a0, x);
        y2 = allpass(D, w2, &p2, a0, x);
        y3 = lowpass(D, w3, &p3, M, a, b, v, x);

        fprintf(fp1, "%lf\n", y1);
        fprintf(fp2, "%lf\n", y2);
        fprintf(fp3, "%lf\n", y3);
        }
}
```

Problem 8.12

A typical program is as follows:

```
/* reverb.c - Schroeder's plain/allpass reverb system */

#define  D1   29
#define  D2   37
#define  D3   44
#define  D4   50
#define  D5   27
#define  D6   31

#define  a1   .75
#define  a2   .75
#define  a3   .75
#define  a4   .75
#define  a5   .75
#define  a6   .75

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double plain(), allpass();

void main()
{
    double *w1, *p1;
    double *w2, *p2;
    double *w3, *p3;
    double *w4, *p4;
    double *w5, *p5;
    double *w6, *p6;
    double y1, y2, y3, y4, y5, y, x;
    int Ntot=500, n;

    FILE *fpy;

    fpy = fopen("y.dat", "w");


    w1 = (double *) calloc(D1+1, sizeof(double));
    p1 = w1;

    w2 = (double *) calloc(D2+1, sizeof(double));
```

```
    p2 = w2;

    w3 = (double *) calloc(D3+1, sizeof(double));
    p3 = w3;

    w4 = (double *) calloc(D4+1, sizeof(double));
    p4 = w4;

    w5 = (double *) calloc(D5+1, sizeof(double));
    p5 = w5;

    w6 = (double *) calloc(D6+1, sizeof(double));
    p6 = w6;

    for (n=0; n<Ntot; n++) {

       if(n==0)                        impulse
              x = 1;
       else
              x = 0;

       y1 = plain(D1, w1, &p1, a1, x);
       y2 = plain(D2, w2, &p2, a2, x);
       y3 = plain(D3, w3, &p3, a3, x);
       y4 = plain(D4, w4, &p4, a4, x);

       y = y1 + 0.9*y2 + 0.8*y3 + 0.7*y4;

       y5 = allpass(D5, w5, &p5, a5, y);
       y  = allpass(D6, w6, &p6, a6, y5);

       fprintf(fpy, "%lf\n", y);
       }
}
```

Problem 8.13

The difference equations are:

$$v(n) = y(n-D) - a_1 v(n-1)$$

$$u(n) = b_0 v(n) + b_1 v(n-1)$$

$$y(n) = x(n) + u(n)$$

The corresponding circular version of the sample processing algorithm is

for each input x do:
- $s_D = \text{tap}(D, \mathbf{w}, p, D)$
- $v_0 = s_D - a_1 v_1$
- $u = b_0 v_0 + b_1 v_1$
- $v_1 = v_0$
- $y = x + u$
- $*p = y$
- $\text{cdelay}(D, \mathbf{w}, \&p)$

Problem 8.14

The response to a causal sinusoid will be of the form:

$$e^{j\omega n}u(n) \longrightarrow H(\omega)e^{j\omega n}u(n) + \sum_{i=1}^{D+1} B_i p_i^n u(n)$$

where $B_i = A_i p_i / (p_i - e^{j\omega})$, as shown in Problem 6.29. It follows that the response to a delayed sinusoid will be:

$$e^{j\omega(n-L)}u(n-L) \longrightarrow H(\omega)e^{j\omega(n-L)}u(n-L) + \sum_{i=1}^{D+1} B_i p_i^{n-L} u(n-L)$$

Multiplying both sides by the factor $e^{j\omega L}$ and subtracting from the undelayed result, we find that the response to the finite sinusoid:

$$x(n) = e^{j\omega n}[u(n) - u(n-L)]$$

is

$$y(n) = H(\omega)e^{j\omega n}[u(n) - u(n-L)] + \sum_{i=1}^{D+1} B_i p_i^n [u(n) - e^{j\omega L} p_i^{-L} u(n-L)]$$

Problem 8.15

A typical C program that utilizes the routines `plain` and `lowpass` is given below:

```
/* reverbex.c - plain and lowpass reverberation of sinusoid */
 *
 * run with:
 * D=300, L=150, N=300, w=0.2pi, w=pi
 *
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double plain(), lowpass();

void main(int argc, char **argv)
{
       double *w1, *p1;
       double *w2, *p2;

       double v[2] = {0., 0.};
       double a[2] = {1, -0.5};
       double b[2] = {0.3, 0.15};

       double a0 = 0.75;
       double w, x, y1, y2, sD;
       double pi = 4*atan(1.0);
       int D, L, N, M=1, n;

       FILE *fp1, *fp2;
```

```
        fp1 = fopen("ypl.dat", "w");
        fp2 = fopen("ylp.dat", "w");

        if (argc != 5) {
               puts("usage: reverbex D L N w\n\n"
                    "D  = delay period\n"
                    "L  = time to turn off input\n"
                    "N  = total duration of input \n"
                    "w  = in units of pi \n");
               exit(0);
               }

        D = atoi(argv[1]);
        L = atoi(argv[2]);
        N = atoi(argv[3]);
        w = atof(argv[4]);      in units of pi

        w1 = (double *) calloc(D+1, sizeof(double));
        p1 = w1;
        w2 = (double *) calloc(D+1, sizeof(double));
        p2 = w2;


for (n=0; n<N; n++) {
        if (n < L)                          sinusoid on for L samples
               x = cos(pi*w*n);
        else
               x = 0;

        y1 = plain(D, w1, &p1, a0, x);
        y2 = lowpass(D, w2, &p2, M, a, b, v, x);

        fprintf(fp1, "%.16lf\n", y1);
        fprintf(fp2, "%.16lf\n", y2);
        }
}
```

The analytical expression for $y(n)$ can be obtained as follows. Using the MATLAB function `lprevb.m` given below, we determine the poles p_i and residues A_i of the transfer function:

```
function [p, A] = lprvb(b, a, D)

% returns poles and residues of lowpass reverberator (delay D)
% [p, A] = lprvb(b, a, D)
%
% b=[b(1), b(2)]
% a=[1,   a(2)]

c = [a, zeros(1,D-2), -b];

p=roots(c);

for i=1:D+1,
       A(i) = 1 + a(2) / p(i);
       for j=1:D+1,
              if j ~= i,
                     A(i) = A(i) * p(i) / (p(i) - p(j));
              end
       end
```



```
       end
end
```

Then, the following MATLAB function calculates the residues B_i and computes N samples of the output signal $y(n)$:

```
function [y, p, A] = yon(N, L, w, b, a, D)

% calculate y(n) for a finite-duration sinusoidal input.

[p, A] = lprvb(b, a, D);

z = exp(j * w);
H = 1 / (1 - z^(-D) * (b(1) + z^(-1)*b(2)) / (1 + a(2)*z^(-1)));

for i=1:D+1,
       B(i) = A(i) * p(i) / (p(i) - z);
end

for n=0:N-1,
       if (n<L),
              y(n+1) = H * z^n;
              for i=1:D+1,
                     y(n+1) = y(n+1) + B(i) * p(i)^n;
              end
       else
              y(n+1) = 0;
              for i=1:D+1,
                     y(n+1) = y(n+1) + B(i) * p(i)^n * (1 - z^L / p(i)^L);
              end
       end
end
```

Problem 8.16

A typical C program is as follows:

```
/* karplus.c - Karplus-Strong String Algorithm */
 *
 * run with:
 * D=25, Ntot=500, iseed=1234567
 * D=50, Ntot=500, iseed=1234567
 *
/*

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double ran(), lowpass();

void main(int argc, char **argv)
{
       double *w, *p;

       double v[2] = {0., 0.};
       double a[2] = {1, 0.0};
```

```
        double b[2] = {0.5, 0.5};

        double x, y;
        int D, Ntot, M=1, n;
        long iseed;

        FILE *fpy;

        fpy = fopen("y.dat", "w");

        if (argc != 4) {
               fprintf(stderr, "usage: karplus D Ntot iseed");
               exit(0);
               }

        D     = atoi(argv[1]);
        Ntot  = atoi(argv[2]);
        iseed = atol(argv[2]);

        w = (double *) calloc(D+1, sizeof(double));

        for (n=0; n<=D; n++)                                    initialize
               w[n] = ran(&iseed) - 0.5;

        p = w;

    for (n=0; n<Ntot; n++) {
        y = lowpass(D, w, &p, M, a, b, v, 0.0);           x=0
        fprintf(fpy, "%.16lf\n", y);
        }
}
```

The output signals corresponding to $D = 25$ and $D = 50$ are shown in Fig. P8.8. Note how the initial random numbers get smoothed at each D-cycle resulting in a decaying quasi-periodic signal tuned to the frequency $f = f_s/D$.

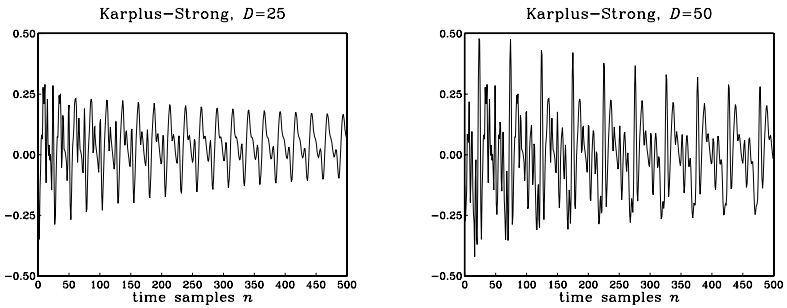


Fig. P8.8 Karplus-Strong string algorithm outputs.

Problem 8.17

A block diagram realization is given in Fig. P8.9. The difference equations are the system:

$$w(n) = aw(n-D)+x(n)$$
$$y(n) = cx(n)+bw(n-D)$$

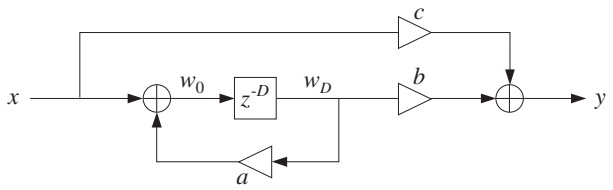


Fig. P8.9 Prototypical delay effect.

The sample processing algorithm is in its linear and circular buffer forms:

for each input x do:
$y = cx + bw_D$
$w_0 = aw_D + x$
delay$(D, \mathbf{w})$

for each input x do:
$s_D = \text{tap}(D, \mathbf{w}, p, D)$
$y = cx + bs_D$
$*p = as_D + x$
cdelay$(D, \mathbf{w}, \&p)$

Problem 8.18

The block diagrams and sample processing algorithms are essentially given in Section 8.2.4. The C routines are:

```
/* plaindel.c - plain reverberating delay */

double tap();
void cdelay();

double plaindel(D, w, p, a, x)              usage: y=plaindel(D,w,&p,a,x);
double *w, **p, a, x;                       p is passed by address
int D;
{
       double y;

       y = tap(D, w, *p, D);                D-th tap delay output
       **p = x + a * y;                     delay input
       cdelay(D, w, p);                     update delay line

       return y;
}

/* lpdel.c - lowpass reverberating delay */

double tap(), can();
void cdelay();

double lpdel(D, w, p, M, a, b, v, x)
double *w, **p, *a, *b, *v, x;
```

```
int D;
{
        double y;

        y = tap(D, w, *p, D);                   D-th tap delay output
        **p = x + can(M, a, M, b, v, y);        delay input
        cdelay(D, w, p);                        update delay line

        return y;
}
```

The routines differ from `plain` and `lowpass` in that the filter outputs are taken after the delay z^{-D}, instead of before.

Problem 8.19

Typical C code for this problem is:

```
for (n=0; n<Ntot; n++) {
    if (n<L)            /* use if (n<1) for impulse response */
            x = 1;
    else
            x = 0;

    x1 = plaindel(D1, w1, &p1, a0, x);                  /* plain */
    x2 = plaindel(D2, w2, &p2, a0, x1);

/*  x1 = lpdel(D1, w1, &p1, 1, a, b, v1, x);  */        /* lowpass */
/*  x2 = lpdel(D2, w2, &p2, 1, a, b, v2, x1); */

    y = b0 * x + b1 * x1 + b2 * x2;                     /* output */

    fprintf(fpx, "%.16lf\n", x);
    fprintf(fpy, "%.16lf\n", y);
    }
```

where the initializations are:

```
    double x, x1, x2, y, *w1, *w2, *p1, *p2;

    double v1[2] = {0., 0.};                    /* state of LP filter 1 */
    double v2[2] = {0., 0.};                    /* state of LP filter 2 */
    double a[2] = {1, -0.5};                    /* G(z) denominator */
    double b[2] = {0.3, 0.15};                  /* G(z) numerator */
    double b0=1, b1=0.8, b2=0.6                 /* feed-forward coefficients */
    double a0 = 0.75;                           /* feedback for plain delay */

    w1 = (double *) calloc(D1+1, sizeof(double));  p1 = w1;
    w2 = (double *) calloc(D2+1, sizeof(double));  p2 = w2;
```

Typical output is as in Fig. P8.10.

Problem 8.20

The sample processing routine is as follows. It can easily be generalized to more than two multi-tap delay segments:

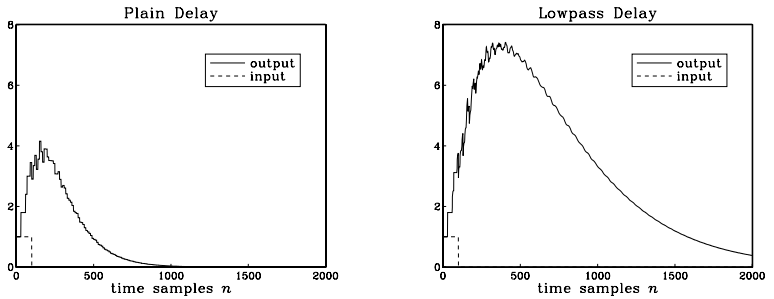


Fig. P8.10 Multi-delay effects processor outputs.

```
/* mlttap.c - multi-tap delay line */

double tap();
void cdelay();

double mlttap(D1, D2, b, a, w, p, x)
int D1, D2;
double b[3], a[3], *w, **p, x;
{
       double s[3], y;

       s[1] = tap(D1+D2, w, *p, D1);
       s[2] = tap(D1+D2, w, *p, D1+D2);
       s[0] = x + a[1] * s[1] + a[2] * s[2];
       y = b[0] * x + b[1] * s[1] + b[2] * s[2];
       **p = s[0];
       cdelay(D1+D2, w, p);

       return y;
}
```

Typical output is shown in Fig. P8.11 and is obtained by the code segment:

```
for (n=0; n<Ntot; n++) {
    if (n < L)                              /* use L=1 or L=200 */
          x = 1;
    else
          x = 0;

    y = mlttap(D1, D2, b, a, w, &p, x);   /* p passed by reference */

    fprintf(fpx, "%.8lf\n", x);
    fprintf(fpy, "%.8lf\n", y);
    }
```

Problem 8.21

From the pole equation, we have:

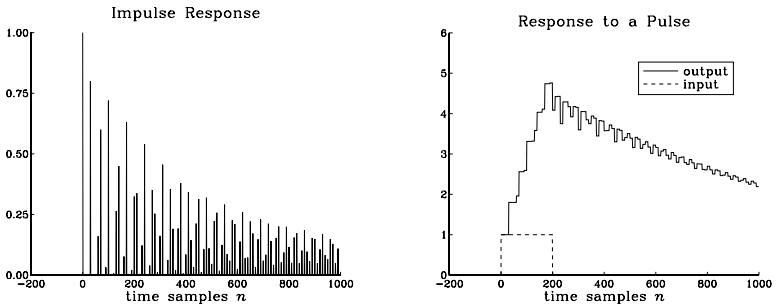


Fig. P8.11 Multi-Tap Delay Line.

$$\begin{aligned} z^{D_1+D_2} &= a_1 z^{D_2} + a_2 \quad \Rightarrow \\ |z|^{D_1+D_2} &= |a_1 z^{D_2} + a_2| \le |a_1||z|^{D_2} + |a_2| \end{aligned} \tag{P8.1}$$

If a pole had $|z| \ge 1$, then using the inequality $|a_1| + |a_2| < 1$, we would have:

$$|z|^{D_1+D_2} \ge |z|^{D_2} > |a_1||z|^{D_2} + |a_2||z|^{D_2} \ge |a_1||z|^{D_2} + |a_2|$$

which contradicts Eq. (P8.1). Thus, all poles must have $|z| < 1$.

Problem 8.22

Working in the z-domain and denoting by $W_L(z)$ and $W_R(z)$ the inputs to the delays z^{-L} and z^{-R}, the outputs of these delays will be $z^{-L}W_L(z)$ and $z^{-R}W_R(z)$. Thus, the I/O equations of the block diagram will be:

$$\begin{aligned} Y_L(z) &= c_L X_L(z) + z^{-L} W_L(z) \\ W_L(z) &= z^{-L} G_L(z) W_L(z) + b_L X_L(z) + d_R z^{-R} W_R(z) \\ Y_R(z) &= c_R X_R(z) + z^{-R} W_R(z) \\ W_R(z) &= z^{-R} G_L(z) W_R(z) + b_R X_R(z) + d_L z^{-L} W_L(z) \end{aligned}$$

Solving the first and third for the Ws, we get:

$$W_L(z) = z^L \big(Y_L(z) - c_L X_L(z)\big), \qquad W_R(z) = z^R \big(Y_R(z) - c_R X_R(z)\big)$$

Inserting them into the second and fourth, we get the system:

$$\begin{aligned} \big(1 - z^{-L} G_L(z)\big)\big(Y_L(z) - c_L X_L(z)\big) - d_R z^{-L}\big(Y_R(z) - c_R X_R(z)\big) &= b_L z^{-L} X_L(z) \\ \big(1 - z^{-R} G_R(z)\big)\big(Y_R(z) - c_R X_R(z)\big) - d_L z^{-R}\big(Y_L(z) - c_L X_L(z)\big) &= b_R z^{-R} X_R(z) \end{aligned}$$

Solving for $Y_L(z), Y_R(z)$ in terms of $X_L(z), X_R(z)$, we get:

$$\begin{aligned} Y_L(z) &= H_{LL}(z) X_L(z) + H_{LR}(z) X_R(z) \\ Y_R(z) &= H_{RL}(z) X_L(z) + H_{RR}(z) X_R(z) \end{aligned}$$

where

$$\begin{aligned} H_{LL}(z) &= c_L + \frac{1}{D(z)}\big(1 - z^{-R} G_R(z)\big) b_L z^{-L} \\ H_{LR}(z) &= \frac{1}{D(z)} d_R b_R z^{-L} z^{-R} \\ H_{RL}(z) &= \frac{1}{D(z)} d_L b_L z^{-R} z^{-L} \\ H_{RR}(z) &= c_R + \frac{1}{D(z)}\big(1 - z^{-L} G_L(z)\big) b_R z^{-R} \end{aligned}$$

with

$$D(z) = \big(1 - z^{-R} G_R(z)\big)\big(1 - z^{-L} G_L(z)\big) - d_L d_R z^{-L} z^{-R}$$

For the special case when the cross couplings are $d_L \neq 0$ and $d_R = 0$, we have:

$$D(z) = \big(1 - z^{-R} G_R(z)\big)\big(1 - z^{-L} G_L(z)\big)$$

and therefore, the transfer functions simplify to:

$$\begin{aligned} H_{LL}(z) &= c_L + \frac{b_L z^{-L}}{1 - z^{-L} G_L(z)} \\ H_{LR}(z) &= 0 \\ H_{RL}(z) &= \frac{d_L b_L z^{-L} z^{-R}}{\big(1 - z^{-R} G_R(z)\big)\big(1 - z^{-L} G_L(z)\big)} \\ H_{RR}(z) &= c_R + \frac{b_R z^{-R}}{1 - z^{-R} G_R(z)} \end{aligned}$$

The corresponding I/O system is then:

$$\begin{aligned} Y_L(z) &= H_{LL}(z) X_L(z) \\ Y_R(z) &= H_{RL}(z) X_L(z) + H_{RR}(z) X_R(z) \end{aligned}$$

which means that the left channel responds only to the left input (it does not receive any cross-feedback from the right channel), but the right channel responds to both the right input and the cross feedback from the left output. If both cross-feedbacks are zero, $d_L = d_R = 0$, then the left and right channels decouple completely, responding independently to their respective inputs. For the case $G_L(z) = a_L$, $G_R(z) = a_R$, the sample processing algorithm is straightforward. Denote the left and right delay-line buffers by

$$\begin{aligned} \mathbf{w}_L &= [w_{L0}, w_{L1}, \ldots, w_{LL}] = (L+1)\text{-dimensional} \\ \mathbf{w}_R &= [w_{R0}, w_{R1}, \ldots, w_{RR}] = (R+1)\text{-dimensional} \end{aligned}$$

Let p_L and p_R be the circular pointers circulating over them, and let the corresponding delay outputs be s_L and s_R. Then, the computational sample processing algorithm will be:

for each input pair $\{x_L, x_R\}$ *do:*

$s_L = \text{tap}(L, \mathbf{w}_L, p_L, L)$
$s_R = \text{tap}(R, \mathbf{w}_R, p_R, R)$
$y_L = c_L x_L + s_L$
$y_R = c_R x_R + s_R$
$*p_L = b_L x_L + a_L s_L + d_R s_R$
$*p_R = b_R x_R + a_R s_R + d_L s_L$
$\text{cdelay}(L, \mathbf{w}_L, \&p_L)$
$\text{cdelay}(R, \mathbf{w}_R, \&p_R)$

Problem 8.23

The sample processing algorithm is implemented by the C code segment:

```
int L = 30, R = 70;
double *wL, *wR, *pL, *pR;

wL = (double *) calloc(L+1, sizeof(double));  pL = wL;
wR = (double *) calloc(R+1, sizeof(double));  pR = wR;

for (n=0; n<Ntot; n++) {
    if (n < P)
          xL = 1;
    else
          xL = 0;

    sL = tap(L, wL, pL, L);
    sR = tap(R, wR, pR, R);
    yL = cL * xL + sL;
    yR = cR * xR + sR;                          /* use xR = 0 */
    *pL = bL * xL + aL * sL + dR * sR;
    *pR = bR * xR + aR * sR + dL * sL;
    cdelay(L, wL, &pL);
    cdelay(R, wR, &pR);

    fprintf(fp1, "%.16lf\n", yL);
    fprintf(fp2, "%.16lf\n", yR);
    }
```

Figures P8.12 and P8.13 show typical outputs. The left output does not start until $n = L$, therefore, the first right output will begin at $n = L + R$. That output will be fed back into the left and will come out at $n = 2L + R$. Similarly, the second left pulse at $n = 2L$ will appear on the right at $n = 2L + R$, and so on.

Problem 8.24

The sample processing algorithm is implemented by the C code segment:

```
if (L > 1) {                                        /* L-point smoother */
       h = (double *) calloc(L, sizeof(double));
       v = (double *) calloc(L, sizeof(double));
       for (n=0; n<L; n++)
             h[n] = 1.0 / L;
```

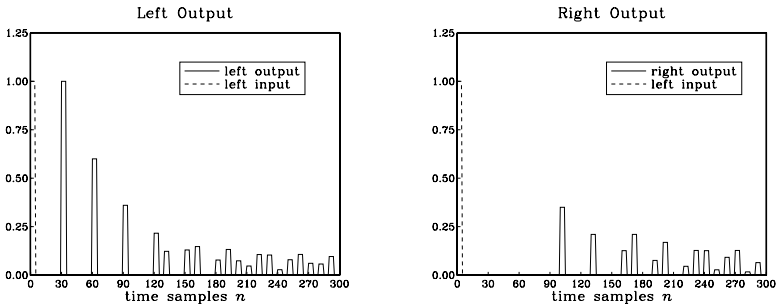



Fig. P8.12 Stereo delays, with $d_L = d_R = 0.3$.

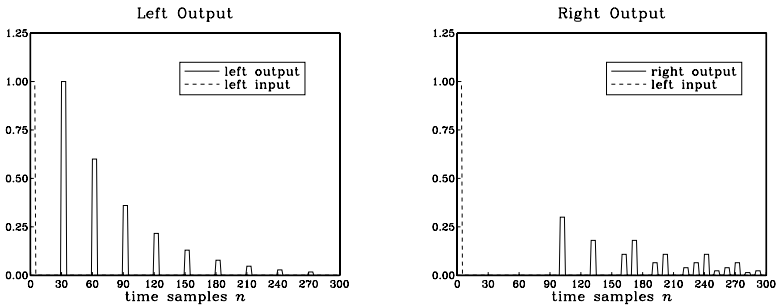


Fig. P8.13 Stereo delays, with $d_L = 0.3$, $d_R = 0$.

```
      }

for (n=0; n<N; n++) {                               /* use N = 600 */
      if (n < N/3)                                  /* generate input */
            x = A1 * cos(w0 * n);                   /* use w0 = 0.15 * pi */
      else if (n < 2*N/3)
            x = A2 * cos(w0 * n);
      else
            x = A3 * cos(w0 * n);

      c = a * c1 + (1-a) * fabs(x);                 /* control signal */
      c1 = c;                                       /* update delay */

      g = f(rho, c/c0);                             /* gain function */

      if (L > 1)
            G = fir(L-1, h, v, g);                  /* gain smoothing */
      else
            G = g;                                  /* no smoothing */

      y = G * x;                                    /* compressor output */
```

```
        fprintf(fpx, "%.16lf\n", x);                    /* save input */
        fprintf(fpy, "%.16lf\n", y);                    /* save output */
        fprintf(fpc, "%.16lf\n", c);                    /* save control */
        fprintf(fpg, "%.16lf\n", G);                    /* save gain */
        }
```

where the compressor function is defined by $f(x) = x^{\rho-1}$, for $x \geq 1$:

```
/* compressor function with 1:rho compressor ratio */

#include <math.h>

double f(rho, x)
double rho, x;
{
        if (x >= 1)
                return pow(x, rho-1);
        else
                return 1;
}
```

The compressing action takes place only above the threshold. For an expander, the gain function is defined as follows, where now $\rho > 1$, but it takes effect only below the threshold:

```
/* expander gain function with 1:rho expander ratio */

#include <math.h>

double f(rho, x)
double rho, x;
{
        if (x <= 1)
                return pow(x, rho-1);
        else
                return 1;
}
```

Problem 8.25

Increasing the threshold to c_0 will cause both A_1 and A_3 to be attenuated. However, A_3 because it is further from c_0 than in the case of Fig. 8.2.38, it will be attenuated more. Fig. P8.14 shows the results.
The noise gate with threshold $c_0 = 1.5$, will suppress both A_1 and A_3, but A_3 by much more, for the same reasons explained above. Fig. P8.15 shows the results.

Problem 8.26

Assuming the causal sequence $\{x(0), x(1), \ldots\}$ is zero-mean white, then the terms in the convolutional sum will be mutually uncorrelated and zero-mean:

$$y(n) = h_0 x(n) + h_1 x(n-1) + \cdots + h(n)x(0)$$

Thus, the output will have zero mean and its variance will be the sum of the variances of the successive terms:

$$\sigma_y^2 = E[y(n)^2] = h_0^2\sigma_x^2 + h_1^2\sigma_x^2 + \cdots h_n^2\sigma_x^2 = \sigma_x^2 \sum_{n=0}^{n} h_m^2$$

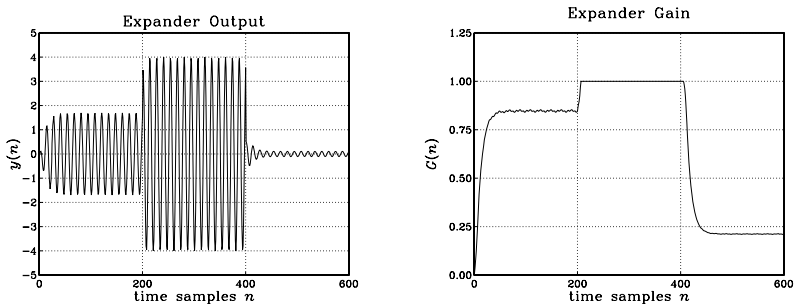


Fig. P8.14 Expander output and gain with increased threshold.

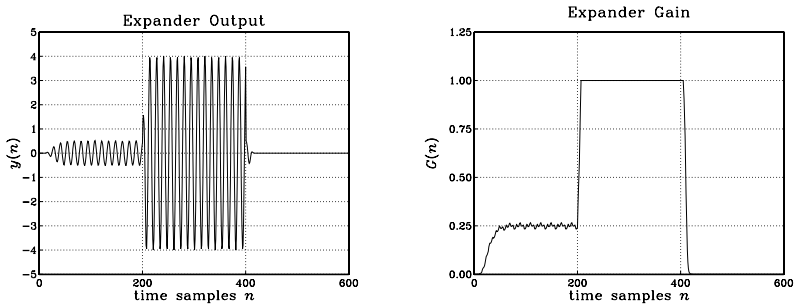


Fig. P8.15 Noise gate output and gain with increased threshold.

Problem 8.27

Assuming a partial fraction expansion of the form:

$$H(z) = \sum_{i=1}^{M} \frac{A_i}{1 - p_i z^{-1}}$$

we have for the impulse response:

$$h_n = \sum_{i=1}^{M} A_i p_i^n u(n)$$

It follows that we have the bound:

$$|h_n| = |\sum_{i=1}^{M} A_i p_i^n| \leq \sum_{i=1}^{M} |A_i||p_i|^n \leq \sum_{i=1}^{M} |A_i||p_{\max}|^n = C|p_{\max}|^n$$

where $C = \sum_{i=1}^{M} |A_i|$ and we used the inequality $|p_i| \leq |p_{\max}|$. The NRR inequality follows now by squaring and summing the above inequality for $|h_n|$ and applying the geometric series on the right-hand side.

Problem 8.28

For ideal filter shapes, the NRR is computed most conveniently using its frequency response definition:

$$NRR = \frac{\sigma_{y_v}^2}{\sigma_v^2} = \int_{-\pi}^{\pi} |H(\omega)|^2 \frac{d\omega}{2\pi}$$

For a bandpass filter, $H(\omega)$ is non-zero only over the interval $\omega_a \le |\omega| \le \omega_b$, where it is unity. Thus,

$$NRR = \frac{\sigma_{y_v}^2}{\sigma_v^2} = \int_{-\omega_b}^{-\omega_a} 1 \cdot \frac{d\omega}{2\pi} + \int_{\omega_a}^{\omega_b} 1 \cdot \frac{d\omega}{2\pi} = \frac{\omega_b - \omega_a}{\pi}$$

Problem 8.29

The following program is an implementation:

```
/* smooth1.c - 1st-order exponential smoother */

#include <stdio.h>
#include <math.h>

void main(int argc, char **argv)
{
       double a, x, y, w1 = 0;

       if (argc != 2) {
              fprintf(stderr, "usage: smooth1 a <x.dat >y.dat\n");
              exit(0);
              }

       a = atof(argv[1]);
       fprintf(stderr, "%lf\n", a);

       while(scanf("%lf", &x) != EOF) {
              y = a * w1 + (1-a) * x;
              w1 = y;
              printf("%lf\n", y);
              }
}
```

Problem 8.30

The mean of $x(n)$ is $m_x = s = 5$. The mean of $y(n)$ is the same because it is preserved through the filter. Indeed, taking means of both sides of

$$y(n) = ay(n-1) + (1-a)x(n)$$

gives

$$m_y = am_y + (1-a)m_x \qquad \Rightarrow \qquad m_y = m_x$$

The mean-square values may be determine recursively as follows: If we define

$$s_n^2 = \frac{1}{n}\sum_{k=0}^{n-1}(x_k - m_x)^2$$

then we have the recursion:

$$s_{n+1}^2 = s_n^2 + \frac{1}{n+1}\left((x_n - m_x)^2 - s_n^2\right)$$

initialized with $s_0^2 = 0$. The following program will calculate the input and output signals and the experimental mean-square values. Any discrepancies are due to the filter transients and they disappear with increasing L:

```
/* smoothex.c - filtering with 1st order smoother */

#include <stdio.h>
#include <math.h>

double gran();

void main(int argc, char **argv)
{
       int n, L;
       long iseed;
       double s=5, x, y, w1, a, m=0, sigma=1;
       double sx2=0, sy2=0;
       FILE *fpx, *fpy;

       if (argc != 4) {
              puts("\nUsage: smoothex a iseed L");
              exit(0);
              }

       a = atof(argv[1]);
       iseed = atol(argv[2]);
       L = atoi(argv[3]);

       fpx = fopen("ttx", "w");
       fpy = fopen("tty", "w");

       for (w1=0, n=0; n<L; n++) {
              x = s + gran(m, sigma, &iseed);
              y = (1 - a) * x + a * w1;
              w1 = y;
              fprintf(fpx, "%lf\n", x);
              fprintf(fpy, "%lf\n", y);
              sx2 += ((x-s)*(x-s) - sx2) / (n+1);
              sy2 += ((y-s)*(y-s) - sy2) / (n+1);
              }

       printf("theoretical  1/NRR = %lf\n", (1+a)/(1-a));
       printf("experimental 1/NRR = %lf\n", sx2/sy2);
}
```

Problem 8.31

The requirement that the high-frequency signal $s(-1)^n$ go through unchanged is that the filter have unity gain at $\omega = \pi$ or $z = -1$. This requires the value for the constant b:

$$\left.\frac{b}{1-az^{-1}}\right|_{z=-1} = \frac{b}{1+a} = 1 \qquad \Rightarrow \qquad b = 1 + a$$

Thus the filter transfer function and impulse response will be:

$$H(z) = \frac{1+a}{1-az^{-1}}, \qquad h(n) = (1+a)a^n u(n)$$

The NRR is:

$$NRR = \sum_{n=0}^{\infty} h(n)^2 = (1+a)^2 \frac{1}{1-a^2} = \frac{1+a}{1-a}$$

Thus, the NRR is magnified if $0 < a < 1$. This can be understood in the frequency domain by the Fig. P8.16.

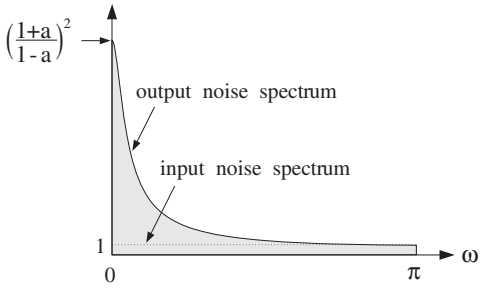


Fig. P8.16 Lowpass filter for extracting high-frequency signal.

The magnitude response is normalized to unity at high frequencies, and therefore it must be larger at low frequencies because it is a lowpass filter. Indeed, its values at DC and AC are:

$$|H(0)|^2 = \left(\frac{1+a}{1-a}\right)^2, \qquad |H(\pi)|^2 = 1$$

Problem 8.32

For an FIR filter of length N, its frequency response will be:

$$H(\omega) = \sum_{n=0}^{N-1} h_n e^{-j\omega n}$$

The condition that the gain is unity at AC is:

$$H(\pi) = \sum_{n=0}^{N-1} h_n (-1)^n = 1$$

where we used the fact $e^{j\pi n} = (-1)^n$. Thus, the NRR must be minimized subject to this AC condition:

$$NRR = \sum_{n=0}^{N-1} h_n^2 = \min, \qquad \text{subject to } \sum_{n=0}^{N-1} h_n (-1)^n = 1$$

This can be solved by Lagrange multipliers or more simply by defining the quantities $g_n = h_n(-1)^n$. Because $g_n^2 = h_n^2$, the problem reduces to the following minimization problem:

$$NRR = \sum_{n=0}^{N-1} g_n^2 = \min, \qquad \text{subject to } \sum_{n=0}^{N-1} g_n = 1$$

whose solution was found in Example 8.3.4 to be $g_n = 1/N$, $n = 0, 1, \ldots, N-1$. Thus, the solution of the highpass problem will be:

$$h_n = (-1)^n g_n = \frac{1}{N}(-1)^n, \qquad n = 0, 1, \ldots, N-1$$

The Lagrange multiplier method is as follows. Introducing a Lagrange multiplier, say λ, that enforces the constraint, we modify the NRR to be minimized into the effective performance index:

$$\mathcal{J} = \sum_{n=0}^{N-1} h_n^2 + 2\lambda\left(1 - \sum_{n=0}^{N-1} h_n (-1)^n\right)$$

The minimization condition with respect to the unknowns h_n are:

$$\frac{\partial \mathcal{J}}{\partial h_n} = 2h_n - 2\lambda(-1)^n = 0, \qquad n = 0, 1, \ldots, N-1$$

This gives $h_n = \lambda(-1)^n$, and the AC constraint then fixes λ to be:

$$\sum_{n=0}^{N-1} h_n (-1)^n = \lambda \sum_{n=0}^{N-1} (-1)^{2n} = \lambda N = 1 \quad \Rightarrow \quad \lambda = \frac{1}{N}$$

Problem 8.33

Expand the transfer function into partial fractions:

$$H(z) = \frac{G}{(1-pz^{-1})(1-p^*z^{-1})} = \frac{A}{1-pz^{-1}} + \frac{A^*}{1-p^*z^{-1}}$$

where

$$A = \frac{G}{1-p^*/p} = \frac{Gp}{p-p^*} = -\frac{jGe^{j\omega_0}}{2\sin\omega_0}$$

The corresponding impulse response and NRR will be then

$$h_n = Ap^n + a^*p^{*n}, \qquad n \geq 0$$

$$NRR = \sum_{n=0}^{\infty} h_n^2 = \sum_{n=0}^{\infty} (A^2p^{2n} + A^{*2}p^{*2n} + 2AA^*|p|^{2n})$$

which sums up to

$$NRR = \frac{A^2}{1-p^2} + \frac{A^{*2}}{1-p^{*2}} + \frac{2|A|^2}{1-|p|^2}$$

Inserting $p = Re^{j\omega_0}$, and A as given above, and replacing G^2 by its normalized value $G^2 = (1-R)^2(1-2R\cos(2\omega_0)+R^2)$ gives the desired expression for NRR after some tedious algebra.

Problem 8.34

For the first filter, we find the transfer function:

$$H(z) = \frac{0.0730(1 - z^{-2})}{1 - 1.7633z^{-1} + 0.8541z^{-2}}$$

Its pole radius is $R = \sqrt{a_2} = \sqrt{0.8541} = 0.9242$, which gives the 5% time constant:

$$n_{\text{eff}} = \frac{\ln(0.05)}{\ln(R)} = 38$$

Its NRR can be computed by carrying out a partial fraction expansion of the form:

$$H(z) = B + \frac{A}{1 - pz^{-1}} + \frac{A^*}{1 - p^*z^{-1}}$$

which gives the impulse response:

$$h_n = B\delta_n + Ap^n u_n + A^* p^{*n} u_n$$

from which we can calculate the NRR:

$$NRR = \sum_{n=0}^{\infty} h_n^2 = B^2 + \sum_{n=1}^{\infty} (Ap^n + A^* p^{*n})^2$$

$$= B^2 + \sum_{n=1}^{\infty} (A^2 p^{2n} + A^{*2} p^{*2n} + 2|A|^2 |p|^{2n})$$

$$= B^2 + \frac{A^2 p^2}{1 - p^2} + \frac{A^{*2} p^{*2}}{1 - p^{*2}} + \frac{2|A|^2 |p|^2}{1 - |p|^2}$$

The parameters B and A are:

$$B = -\frac{G}{|p|^2}, \qquad A = -\frac{G(1 - p^2)}{p^2 - |p|^2}$$

The numerical value of NRR is $NRR = 1/13.4$. For the second filter, we start with n_{eff} from which we may solve for

$$R = (0.05)^{1/n_{\text{eff}}} = 0.9901$$

where R is related to the 3-dB width by:

$$R^2 = a_2 = 2b - 1 = \frac{1 - \tan(\Delta\omega/2)}{1 + \tan(\Delta\omega/2)}$$

which gives $\Delta\omega = 0.0064\pi$. Then, the design equations give:

$$H(z) = \frac{0.0099(1 - z^{-2})}{1 - 1.8833z^{-1} + 0.9802z^{-2}}$$

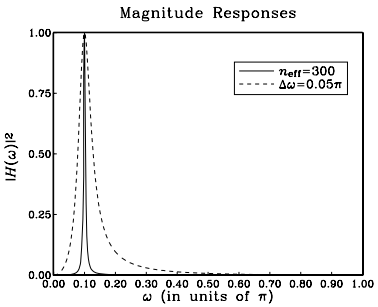
and $NRR = 1/101.1$. The required graphs are shown in Figs. P8.17 and P8.18.

Problem 8.35

For $Q = 6$, the transfer function is designed using Eqs. (8.2.22) and (8.2.23):

$$H(z) = 0.96953 \frac{1 - 1.85955z^{-1} + z^{-2}}{1 - 1.80289z^{-1} + 0.93906z^{-2}}$$

Its 1% time constant is $n_{\text{eff}} = 146$. For $Q = 60$, the filter is identical to that of Example 8.3.8. Figures P8.19–P8.21 show the input signal, filter responses, and output signals.



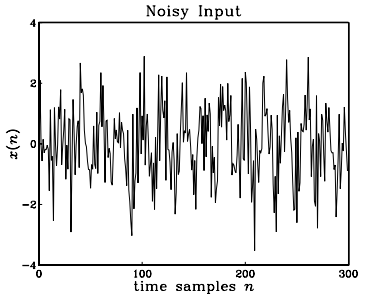
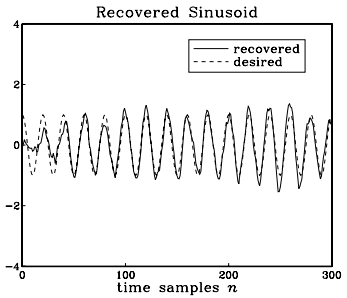


Fig. P8.17 Magnitude responses and noisy input of Problem 8.34.



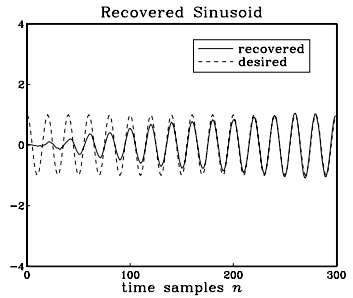
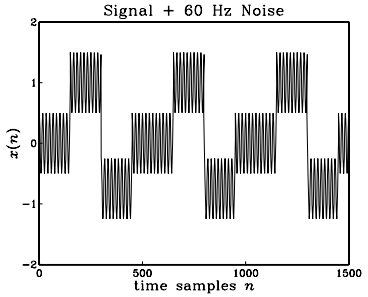


Fig. P8.18 Filter outputs of Problem 8.34.



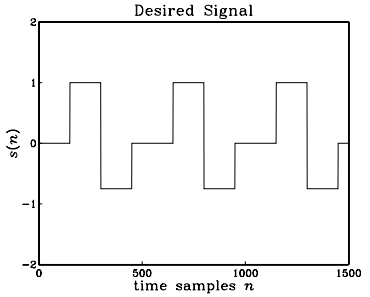


Fig. P8.19 Noisy input and desired signal of Problem 8.35.

Problem 8.36

The designed filter has the form:

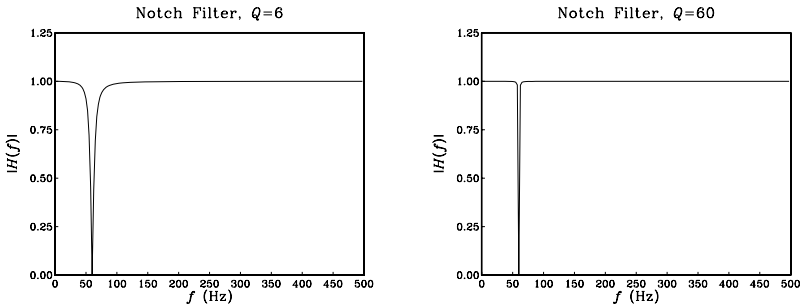


Fig. P8.20 Magnitude responses, for $Q = 6$ and $Q = 60$. Problem 8.35.

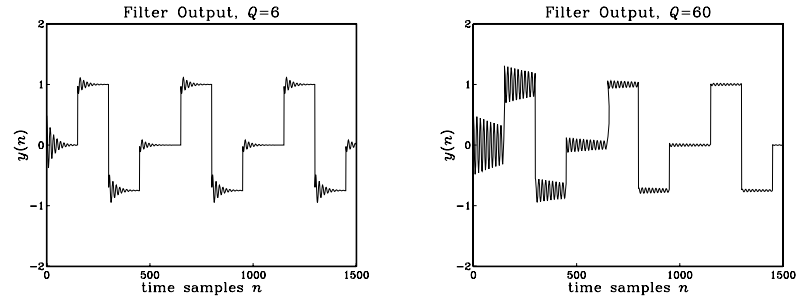


Fig. P8.21 Filter outputs, for $Q = 6$ and $Q = 60$. Problem 8.35.

$$H(z) = b\frac{1 - z^{-D}}{1 - az^{-D}}$$

The time constant is computed as:

$$n_{\text{eff}} = D\frac{\ln \epsilon}{\ln a}$$

where $\epsilon = 0.01$ for the 1% time constant. The filter parameters are in the four cases:

Q	a	b	n_{eff}
80	0.961481	0.980741	1172.4
200	0.984414	0.992207	2931.6
400	0.992177	0.996088	5863.7
800	0.996081	0.998040	11727.8

The relevant graphs are shown in Figs. P8.22–P8.26.

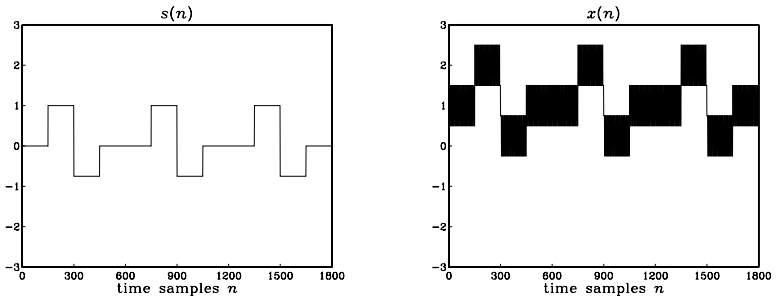


Fig. P8.22 Desired signal and noisy input. Problem 8.36.

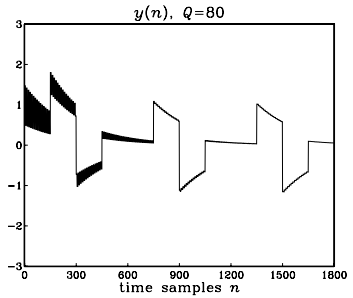


Fig. P8.23 Filtered output $y(n)$ when $Q = 80$. Problem 8.36.

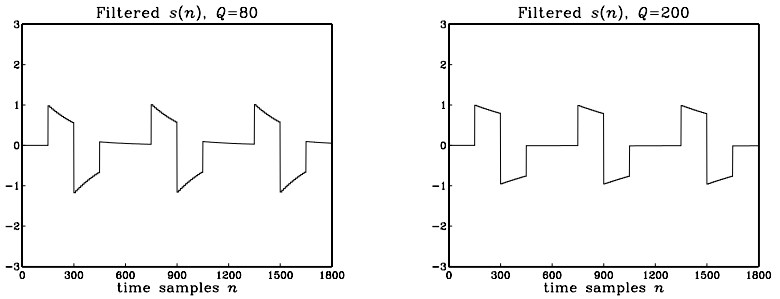


Fig. P8.24 Filtered $s(n)$ for $Q = 80$ and $Q = 200$. Problem 8.36.

Problem 8.37

The following MATLAB m-file designs the 60 Hz notch filter, generates the noisy ECG and filters it, and computes the filter's magnitude response:

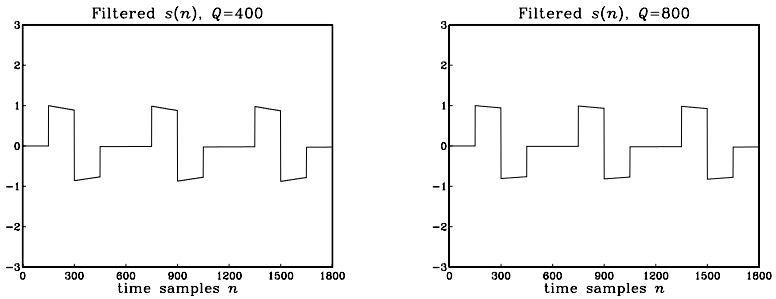


Fig. P8.25 Filtered $s(n)$ for $Q = 400$ and $Q = 800$. Problem 8.36.

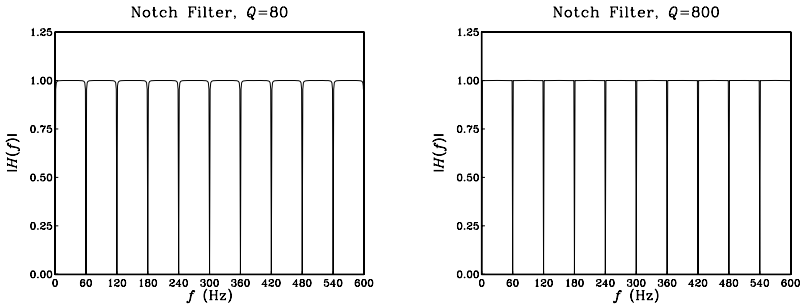


Fig. P8.26 Magnitude responses for $Q = 80$ and $Q = 800$. Problem 8.36.

```
% ecgex1.m - simulated ECG + 60Hz noise example
%
% assume 2 beats/sec => 0.5 sec/beat => 500 samples/beat => 1000 samples/sec
% f0=60; Q=60; df=f0/Q;
%
% compute notch filter coefficients
% generate length-500 ecg by calling x0=ecg(500); normalize it to 1.
% replicate three beats x=[x0,x0,x0] and smooth them with x=sgfilt(0, 5, x);
% add 80 percent 60-Hz noise
% and filter it yv=filter(B,A,xv)
% finally, compute filter's magnitude response.

f0 = 60/1000; Q = 60; df = f0/Q;
w0 = 2 * pi * f0; dw = 2 * pi * df;

b = 1 / (1+tan(dw/2));
B = b * [1, -2*cos(w0), 1];
A = [1, B(2), 2*b-1];

x0 = ecg(500);
x = [x0, x0, x0];
x = sgfilt(0, 5, x); m = max(x); x = x/m;
```



```
n = 0:1499;

xv = x + 0.5 * cos(w0 * n);
yv = filter(B, A, xv);

w = (0:499)*pi/500;  z = exp(-i*w);
h = abs((B(1) + B(2)*z + B(3)*z.^2) ./ (1 + A(2)*z + A(3)*z.^2));
```

Problem 8.38

The following MATLAB m-file designs the 60 Hz comb filter, generates the noisy ECG and filters it, and computes the filter's magnitude response:

```
% ecgex2.m - simulated ECG + 60Hz square-wave noise example
%
% assume 1 beats/sec => 1 sec/beat => 600 samples/beat => 600 samples/sec
%
% compute notch filter coefficients
% generate length-600 ecg by calling x0=ecg(600); normalize it to 1.
% replicate three beats x=[x0,x0,x0] and smooth them with x=sgfilt(0, 9, x);
% add shifted square wave
% and filter it yv=filter(B,A,xv)
% finally, compute filter's magnitude response.

fs = 600;  f0 = 60;  df = 0.75;

w0 = 2 * pi * f0 / fs;
dw = 2 * pi * df / fs;

beta = tan(dw * 10 / 4);                          % D=10 here
a  = (1 - beta) / (1 + beta);
b  = (1 + a) / 2;

B = b * [1 0 0 0 0 0 0 0 0 0 -1];                 % numerator
A = [1 0 0 0 0 0 0 0 0 0 -a];                     % denominator

x0 = ecg(600);
x = [x0, x0];
x = sgfilt(0, 9, x);   m = max(x); x = x/m;

n = 0:1199;

v0 = [1 1 1 1 1 -1 -1 -1 -1 -1];   v = v0;        % one noise period

for k=1:119,                                      % 120 noise periods
      v = [v, v0];
end

v = 2 + v;                                        % noise baseline shift

xv = x + 0.5 * v;                                 % noisy ECG
yv = filter(B, A, xv);

w = (0:500)*pi/500;  z = exp(-i*w);
h = abs(b * (1 - z.^10) ./ (1 - a * z.^10));
```

Problem 8.39

Using partial fractions, we expand

$$H(z) = b\frac{1+z^{-D}}{1-az^{-D}} = A + \frac{B}{1-az^{-D}} = (A+B) + Baz^{-D} + Bz^2z^{-2D} + \cdots$$

where

$$A = -\frac{b}{a}, \quad B = b\left(1+\frac{1}{a}\right), \quad A+B = b$$

Thus, the impulse response is

$$\mathbf{h} = [b, 0, 0, \ldots, 0, Ba, 0, 0, \ldots, 0, Ba^2, 0, 0, \ldots, 0, Ba^3, \ldots]$$

with the nonzero entries separated by $D-1$ zeros. The sum of the squares is

$$\begin{aligned} NRR &= \sum_{n=0}^{\infty} h_n^2 = b^2 + B^2 \sum_{m=1}^{\infty} a^{2m} = b^2 + B^2\frac{a^2}{1-a^2} \\ &= \frac{b^2(1-a^2) + b^2a^2(1+a^{-2})}{1-a^2} = \frac{2b^2(1+a)}{1-a^2} = \frac{2(1-a)^2(1+a)}{4(1-a)(1+a)} \\ &= \frac{1-a}{2} \end{aligned}$$

In the filter design parameter $\beta = \tan(\Delta\omega D/4)$, the tangent has argument:

$$\frac{\Delta\omega D}{4} = \frac{\Delta\omega}{4}\cdot\frac{2\pi}{\omega_1} = \frac{\pi\Delta\omega}{2\omega_1} = \frac{\pi}{2Q}$$

where we used $\omega_1 = 2\pi/D$, or $D = 2\pi/\omega_1$.

Problem 8.40

The following program designs the comb filter, constructs 40 periods of length-50 of the noisy triangular waveform, and filters them using the circular buffer implementation of the canonical realization of the filter:

```
/* combex.c - IIR comb filter with noisy periodic input */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double gran(), tap();
void cdelay(), triang();

void main(int argc, char **argv)
{
    int n, i, N, D;
    long iseed;
    double a, b, pi=4*atan(1.0), dw = 0.0008 * pi;
    double *s, *w, *p;
    double x, y, sD;
    FILE *fpy, *fps, *fpx;

    if (argc != 4) {
        puts("\nUsage: combex D N iseed");
        exit(0);
        }

    D     = atoi(argv[1]);              period
    N     = atoi(argv[2]);              number of periods
    iseed = atol(argv[3]);              initial seed

    fpy = fopen("y.dat", "w");          filter output
    fps = fopen("s.dat", "w");          noise-free input
    fpx = fopen("x.dat", "w");          noisy input

    a  = (1 - tan(D*dw/4)) / (1 + tan(D*dw/4));
    b  = (1 - a) / 2;

    s = (double *) calloc(D, sizeof(double));
    triang(D, s);                       one period of triangular wave

    w = (double *) calloc(D+1, sizeof(double));
    p = w;

    for (i=0; i<N; i++)                 generate N periods
        for (n=0; n<D; n++) {
            x = s[n] + 0.5 * gran(0., 1., &iseed);
            sD = tap(D, w, p, D);
            *p = b * x + a * sD;
            y = (*p) + sD;
            cdelay(D, w, &p);
            fprintf(fps, "%lf\n", s[n]);
            fprintf(fpx, "%lf\n", x);
            fprintf(fpy, "%lf\n", y);
            }
}

/* ------------------------------------------------------- */

/* triang.c - triangular wave */

void triang(D, s)
int D;
double *s;
{
    int i;
    double D4 = D/4.0;

    for (i = 0; i<D4; i++)
       s[i] = i / D4;

    for (i=D4; i<3*D4; i++)
       s[i] = 1 + (D4 - i) / D4;

    for (i=3*D4; i<D; i++)
       s[i] = -1 + (i - 3*D4) / D4;
}
```

Problem 8.41

The complementarity properties may be verified for each pair. For example, for filter 2, we have half-order $M = 8/2 = 4$. Thus, we must verify:

$$H_{LP}(z)+H_{BP}(z)=\frac{1}{16}(1+z^{-2})^2(-1+6z^{-2}-z^{-4})+\frac{1}{16}(1-z^{-2})^4=z^{-4}$$

which follows by expanding out the terms. Fig. P8.27 shows the frequency responses in the four cases. The frequency scale is in MHz and extends only up to about 4.5 MHz. For reference, the Nyquist frequency here is $f_s/2 = 2f_{sc} = 7.159$ MHz, so that the subcarrier frequency shown in the figure is only one-quarter of the Nyquist interval.

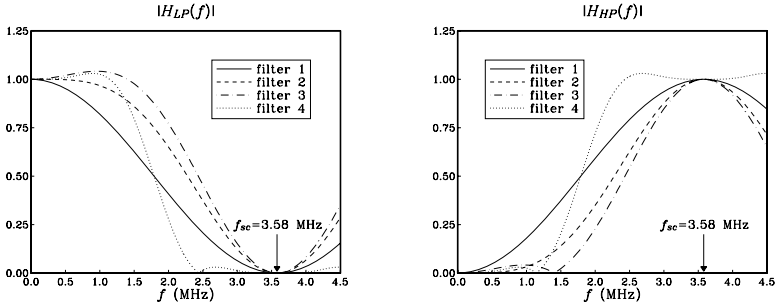


Fig. P8.27 Lowpass/bandpass and highpass/bandstop responses. Problem 8.41.

Problem 8.42

See Table P8.1. Note that the circular pointer p points to the buffer entry $w[q]$, whereas the D-th tap output is the content of $w[q_D]$. At each time instant, the content of $w[q]$ is updated by $w[q] = w[q_D]+x$, where the scaling by N is done afterwards. Table P8.1 shows the results. The advantage of the circular implementation is that only one entry of $\mathbf{w}$, namely, $w[q]$, is changed at each time instant. This is efficient in hardware implementations and for large values of D.

Problem 8.43

The following is a complete C program for performing signal averaging. The inputs to the program are the period D and the number N of periods to be averaged. The input data are taken from `stdin` and the averaged period is output to the `stdout`:

```
/* sigav.c - signal averaging */

#include <stdio.h>
#include <stdlib.h>

void cdelay();
double tap();

void main(int argc, char **argv)
{
    double x, *w, *p;
    int D, N, i, k;

    if (argc != 3) {
       fprintf(stderr, "usage: sigav D N  < x.dat  > y.dat\n");
       fprintf(stderr, "x.dat must contain N periods of length D\n");
       exit(0);
       }

    D = atoi(argv[1]);
    N = atoi(argv[2]);

    w = (double *) calloc(D+1, sizeof(double));
    p = w;

    for (k=0; k<N; k++)
      for (i=0; i<D; i++)
        if (scanf("%lf", &x) != EOF) {      /* keep reading input samples */
           *p = tap(D, w, p, D) + x / N;
           cdelay(D, w, &p);
           }
        else   {
           fprintf(stderr, "Input must have %d periods of length %d\n", N, D);
           exit(0);
           }

    for (i=0; i<D; i++)
      printf("%.12lf\n", tap(D, w, p, D-i));
}
```

n	x	q_D	q	w_0	w_1	w_2	w_3	$y = w_q/4$
0	s_0	3	0	s_0	0	0	0	$s_0/4$
1	s_1	2	3	s_0	0	0	s_1	$s_1/4$
2	s_2	1	2	s_0	0	s_2	s_1	$s_2/4$
3	s_0	0	1	s_0	$2s_0$	s_2	s_1	$2s_0/4$
4	s_1	3	0	$2s_1$	$2s_0$	s_2	s_1	$2s_1/4$
5	s_2	2	3	$2s_1$	$2s_0$	s_2	$2s_2$	$2s_2/4$
6	s_0	1	2	$2s_1$	$2s_0$	$3s_0$	$2s_2$	$3s_0/4$
7	s_1	0	1	$2s_1$	$3s_1$	$3s_0$	$2s_2$	$3s_1/4$
8	s_2	3	0	$3s_2$	$3s_1$	$3s_0$	$2s_2$	$3s_2/4$
9	s_0	2	3	$3s_2$	$3s_1$	$3s_0$	$4s_0$	s_0
10	s_1	1	2	$3s_2$	$3s_1$	$4s_1$	$4s_0$	s_1
11	s_2	0	1	$3s_2$	$4s_2$	$4s_1$	$4s_0$	s_2
12	–	–	–	–	$4s_2$	$4s_1$	$4s_0$	–

Table P8.1 Signal averager with circular delay for $D = 3$, $N = 4$.

Problem 8.44

The following program generates N periods of length D of the noisy signal, and averages these periods using the circular buffer form of the averager. After processing the ND input samples, it writes out the contents of the delay line in reverse order, so that the files `y.dat` and `s.dat` will contain the averaged periods and the noise-free period:

```
/* sigavex.c - signal averaging example */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

double gran(), tap();
void cdelay();

void main(int argc, char **argv)
{
    int n, i, N, D;
    long iseed;
    double *s, x, *w, *p;
    double sD;
    FILE *fpx, *fpy, *fps;

    if (argc != 4) {
        puts("\nUsage: sigavex D N iseed");
        exit(0);
        }

    D     = atoi(argv[1]);              period
    N     = atoi(argv[2]);              number of periods
    iseed = atol(argv[3]);              noise seed

    fpx = fopen("x.dat", "w");
    fpy = fopen("y.dat", "w");
    fps = fopen("s.dat", "w");

    s = (double *) calloc(D, sizeof(double));

    for (i=0; i<D; i++)
        if (i < D/2)
            s[i] = 1;
        else
            s[i] = -1;

    w = (double *) calloc(D+1, sizeof(double));
    p = w;

    for (i=0; i<N; i++)                 accumulate N periods
        for (n=0; n<D; n++) {           each of length D
            x = s[n] + gran(0., 1., &iseed);    noisy input
               sD = tap(D, w, p, D);            D-th tap of averager
               *p = sD + x;                     accumulate x
               cdelay(D, w, &p);                update delay line
               fprintf(fpx, "%.12lf\n", x);     save input sample
               }

    for (n=0; n<D; n++) {               write averaged period
            fprintf(fps, "%lf\n", s[n]);
            fprintf(fpy, "%.12lf\n", tap(D, w, p, D-n)/N);
            }
}
```


Problem 8.45

The following MATLAB function generates the noisy ECG and smoothes it once and twice by Savitzky-Golay filter of length N and smoothing order d:

```
% sgecgex.m - SG filtering of noisy ECG

L=500;

x0 = ecg(L)';                           % generate ECG
x = sgfilt(0, 15, x0);
mx = max(x);
x = x/mx;                               % unity max

rand('normal'); rand('seed', 10033);
v = 0.3 * rand(L, 1);                   % generate noise

xv = x + v;                             % generate noisy ECG

N=11; d=2;
y1 = sgfilt(d, N, xv);                  % first pass through SG
y2 = sgfilt(d, N, y2);                  % second pass
```

The filtering twice through the SG filter improves the smoothing operation but introduces longer transients in the filters. The following figures should be compared with the single-pass cases of Figs. 8.3.33 and 8.3.34.

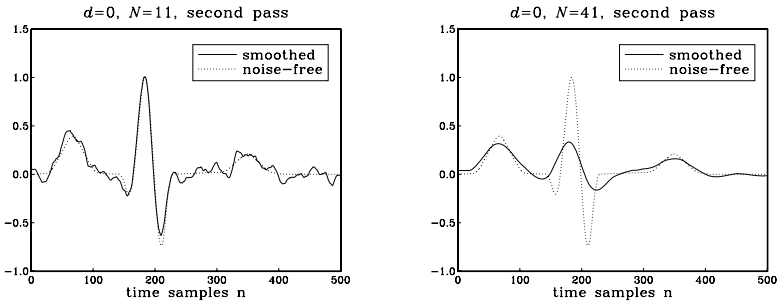


Fig. P8.28 Double SG filtering; $N = 11, 41$ and $d = 0$.

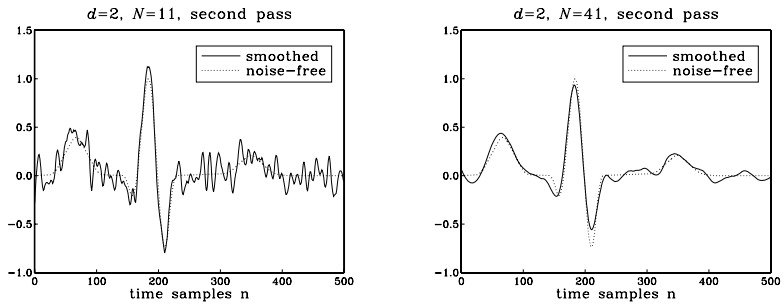


Fig. P8.29 Double SG filtering; $N = 11, 41$ and $d = 2$.

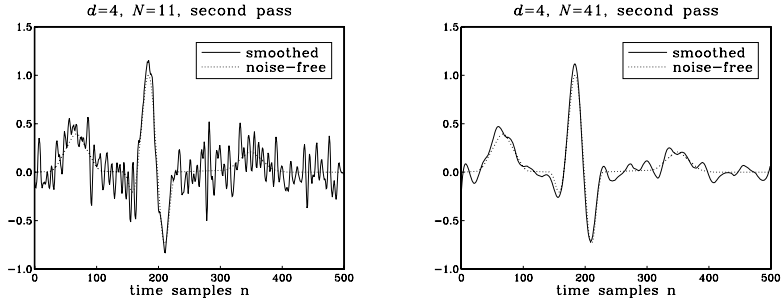


Fig. P8.30 Double SG filtering; $N = 11, 41$ and $d = 4$.

Chapter 9 Problems

Problem 9.1

Use the formula $T_R = LT = L/f_s$ to solve for $L = f_s T_R$, or in words

no. of samples = (no. of samples per second) x (no. of seconds)

Thus, $L = 8$ kHz$\times$128 msec $= 1024$ samples. The frequency spacing is $\Delta f = f_s/N = 8000/256 = 31.25$ Hz. If the DFT is done directly, we would require $L \cdot N = 1024 \cdot 256 = 262144$ multiplications. If the signal is reduced mod-256, then the number of multiplications would be $N \cdot N = 256 \cdot 256 = 65536$. And if the FFT is used, $N \log_2 N/2 = 256 \cdot 8/2 = 1024$.
The cost of performing the mod-256 reduction is $N(M-1) = 256 \cdot 3 = 768$ additions, where $M = L/N = 1024/256 = 4$ is the number of segments. This cost may be added to the costs of FFT or reduced DFT.

Problem 9.2

The sinusoidal signal is something like $x(t) = \cos(2\pi f t)$, where $f = 10$ kHz. Its spectrum will have peaks at $\pm f$. Is there a DFT index k such that the kth DFT frequency f_k equals $\pm f$?

$$\pm f = f_k = \frac{f_s}{N}k \qquad \Rightarrow \qquad k = \pm N\frac{f}{f_s} = \pm 64\frac{10}{80} = \pm 8$$

Thus, we expect to see a peak at $k = 8$ and another one at $k = -8$. Because the DFT $X(k)$ is periodic in k with period N and because we always list the indices in the positive range $k = 0, 1, \cdots, N-1$, we shift the negative index by one period N to get the positive index:

$$-k \to N - k = 64 - 8 = 56$$

Thus, the second peak will be at $k = 56$.

Problem 9.3

The number of samples in one period is given by:

$$D = \frac{f_s}{f} = \frac{40}{5} = 8 \text{ samples}$$

Therefore, if k periods are collected , the total signal length will be

$$N = kD = k\frac{f_s}{f}$$

which gives for $k = 16$, $N = 16 \cdot 8 = 128$ samples. If we perform an N-point DFT, then because of the choice of N, the k-th DFT frequency will coincide with f, that is,

$$f_k = k\frac{f_s}{N} = f$$

Thus, we expect to see a peak at the kth DFT bin. We will also get a peak at the negative index $-k$, which translates mod-N to $N - k$. In summary, we expect to see peaks at the DFT indices:

$$k = 16 \quad \text{and} \quad N - k = 128 - 16 = 112$$

Problem 9.4

The expected DFT index will be at

$$k = N\frac{f}{f_s} = 16 \cdot \frac{18}{8} = 36$$

Because k is greater than $N - 1 = 15$, it may be reduced mod-N down to:

$$k = 36 - 2 \times 16 = 4$$

It corresponds to the DFT frequency $f_k = kf_s/N = 4 \cdot 8/16 = 2$ kHz. This is frequency within the Nyquist interval that is aliased with $f = 18$ kHz; indeed, we have

$$f_{\text{alias}} = f \bmod f_s = 18 \bmod 8 = 18 - 2 \times 8 = 2 \text{ kHz}$$

The -2 kHz or -18 kHz component will be represented by the DFT index $N - k = 16 - 4 = 12$.

Problem 9.5

To be able to resolve a peak, the width of the main lobe of the data window must be equal or smaller than the width of the peak. This gives the condition

$$\frac{f_s}{L} \le \Delta f \qquad \Rightarrow \qquad L \ge \frac{f_s}{\Delta f}$$

or, in our case, $L \ge 8000/20 = 400$ samples. The duration of these 400 samples will be

$$T_R = LT \ge \frac{L}{f_s} = \frac{1}{\Delta f} = \frac{1}{20} = 50 \text{ msec}$$

To avoid wrap-around errors in the inverse FFT, the FFT length must be at least L, i.e., $N \ge L = 400$. The next power of two is $N = 512$. We must pad 112 zeros to the end of the length-400 signal before the FFT is taken.

Problem 9.6

The following MATLAB function generates the time data and computes the required DTFTs with the help of the function `dtft.m`, included below, which replaces the C functions `dtft.c` and `dtftr.c`:

```
% dtftexp.m - rectangular/hamming windowed sinsuoids

L = 200; N = 200;
w0 = 0.1*pi;
w = (0:N-1) * 0.2 * pi / N;              % frequency range
n = 0:(L-1);
wh = 0.54 - 0.46 * cos(2*pi*n/(L-1));    % Hamming window

xrec = cos(w0 * n);
xham = wh .* xrec;                       % windowed data

Xrec = abs(dtft(xrec, w));               % DTFT of rectangular data
Xham = abs(dtft(xham, w));               % DTFT of Hamming data

save xrec.dat xrec /ascii;
save xham.dat xham /ascii;
save frec.dat Xrec /ascii;
save fham.dat Xham /ascii;
```


where

```
% dtft.m - DTFT of a signal at a frequency vector w
%
% X = dtft(x, w);
%
% x = row vector of time samples
% w = row vector of frequencies in rads/sample
% X = row vector of DTFT values
%
% based on and replaces both dtft.c and dtftr.c

function X = dtft(x, w)

[L1, L] = size(x);

z = exp(-j*w);

X = 0;
for n = L-1:-1:0,
       X = x(n+1) + z .* X;
end
```

Problem 9.7

The following MATLAB function generates the time data and computes the required DTFTs:

```
% sinexp.m - 3 sinusoids of length L=10

L=10; N=256;
f1 = 0.2; f2 = 0.25; f3 = 0.3;
n = 0:(L-1);

x = cos(2*pi*f1*n) + cos(2*pi*f2*n) + cos(2*pi*f3*n);
w = 0.54 - 0.46 * cos(2*pi*n/(L-1));
xham = w.*x;

X = abs(fft(x,N));
Xham = abs(fft(xham,N));
```

For the other cases, use $L = 10, 40, 100$.

Problem 9.8

The following C program generates the length-L time signal and computes its N-point DFT, with $L = 10, 20, 100$ and $N = 256$. Instead of the DFT routine `dft.c`, one could use `fft.c`. The 32-point DFT is extracted by keeping one out of every $256/32 = 8$ points of the 256-point DFT.

```
/* dtftex.c - physical vs. computational resolution example */

#include <stdio.h>
#include <cmplx.h>
#include <stdlib.h>

void dtftr();
```

```
main()
{
    int L, N, n, k;
    double *x, pi = 4 * atan(1.0);
    double f1 = 0.20, f2 = 0.25, f3 = 0.30;
    complex *X;
    FILE *fpdtft;

    fpdtft= fopen("dtft.dat", "w");              DTFT file

    printf("enter L, N: ");
    scanf("%d %d", &L, &N);

    x =  (double *) calloc(L, sizeof(double));
    X = (complex *) calloc(N, sizeof(complex));

    for (n=0; n<L; n++)
        x[n] = cos(2*pi*f1*n) + cos(2*pi*f2*n) + cos(2*pi*f3*n);

    dft(L, x, N, X);                             N-point DFT

    for (k=0; k<N; k++)
        fprintf(fpdtft, "%lf\n", cabs(X[k]));
}
```

Problem 9.9

See Example 9.2.1.

Problem 9.10

Using the relationship $k = Nf/fs$, we calculate the group-A and group-B DFT indices:

$$\text{group A} \qquad k_A = 22.3,\ 24.6,\ 27.3,\ 30.1$$

$$\text{group B} \qquad k_B = 38.7,\ 42.8,\ 47.3$$

and their rounded versions:

$$\text{group A:} \qquad k_A = 22,\ 25,\ 27,\ 30$$

$$\text{group B:} \qquad k_B = 39,\ 43,\ 47$$

Because each keypress generates one tone from group A and one from group B, the DTFT will have peaks at the corresponding pair of DFT indices (and their negatives, that is, $N - k_A$ and $N - k_B$.)

Problem 9.11

The time signal and its DTFT are shown in Figs. P9.1 and P9.2, both for the rectangular and Hamming windows.

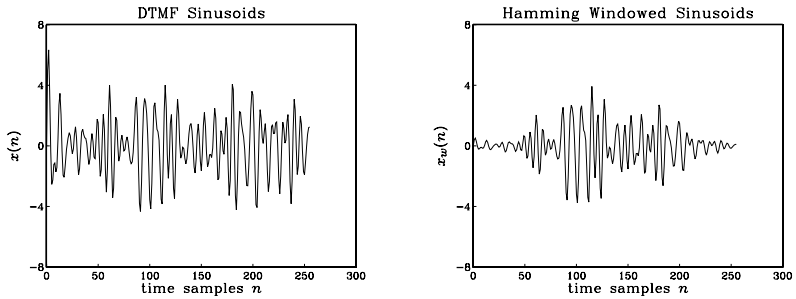


Fig. P9.1 Rectangularly and Hamming windowed DTMF sinusoids.

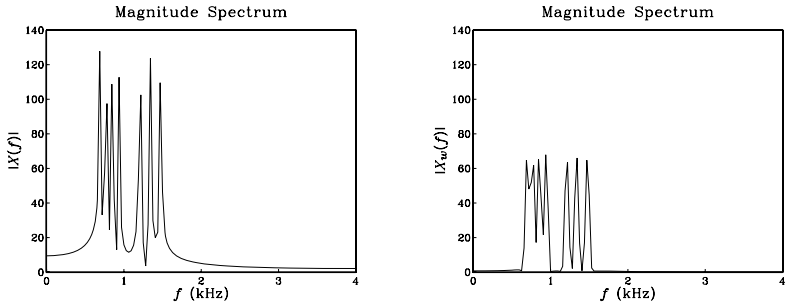


Fig. P9.2 Spectra of DTMF sinusoids.

Problem 9.12

Acting on $\mathbf{x}$ by the 4×8 DFT matrix, we obtain:

$$\mathbf{X} = \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j & 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j & 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 2 \\ 1 \\ 2 \\ 1 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 12 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

The mod-4 wrapped version of $\mathbf{x}$ is $\tilde{\mathbf{x}} = [3,\ 3,\ 3,\ 3]$. Its 4-point DFT is

$$\mathbf{X} = \begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 3 \\ 3 \\ 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 12 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

For the IDFT we use the formula $\text{IDFT}(\mathbf{X}) = [\text{DFT}(\mathbf{X}^*)]^*/N$:

$$\tilde{\mathbf{x}} = \begin{bmatrix} \tilde{x}(0) \\ \tilde{x}(1) \\ \tilde{x}(2) \\ \tilde{x}(3) \end{bmatrix} = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} 12 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \\ 3 \\ 3 \end{bmatrix}$$

Problem 9.13

The 8-point FFT is:

$$\mathbf{x} = \begin{bmatrix} 5 \\ -1 \\ -3 \\ -1 \\ 5 \\ -1 \\ -3 \\ -1 \end{bmatrix} \quad \overset{8-FFT}{\longrightarrow} \quad \mathbf{X} = \begin{bmatrix} 0 \\ 0 \\ 16 \\ 0 \\ 8 \\ 0 \\ 16 \\ 0 \end{bmatrix}$$

It follows that the FFT spectrum will show peaks at the frequencies:

$$X(\omega_2) = X(\omega_6)^* = X(-\omega_2)^* = 16, \qquad X(\omega_4) = 8$$

Problem 9.14

The IFFT is obtained by conjugating **X**, computing its FFT, and dividing by 8. The required FFT of $\mathbf{X}^*$ is:

$$\mathbf{X}^* = \begin{bmatrix} 0 \\ 4 \\ 4j \\ 4 \\ 0 \\ 4 \\ -4j \\ 4 \end{bmatrix} \quad \overset{8-FFT}{\longrightarrow} \quad \begin{bmatrix} 16 \\ 8 \\ 0 \\ -8 \\ -16 \\ 8 \\ 0 \\ -8 \end{bmatrix} \quad \Rightarrow \quad \mathbf{x} = \begin{bmatrix} 2 \\ 1 \\ 0 \\ -1 \\ -2 \\ 1 \\ 0 \\ -1 \end{bmatrix}$$

Using the inverse DFT formula, we express $x(n)$ as a sum of sinusoids:

$$\begin{aligned} x(n) &= \frac{1}{8}\sum_{k=0}^{7} X(k)e^{j\omega_k n} = \frac{1}{8}\left[4e^{j\omega_1 n} - 4je^{j\omega_2 n} + 4e^{j\omega_3 n} + 4e^{j\omega_5 n} + 4je^{j\omega_6 n} + 4e^{j\omega_7 n}\right] \\ &= \frac{1}{2}e^{j\omega_1 n} + \frac{1}{2j}e^{j\omega_2 n} + \frac{1}{2}e^{j\omega_3 n} + \frac{1}{2}e^{-j\omega_3 n} - \frac{1}{2j}e^{-j\omega_2 n} + \frac{1}{2}e^{-j\omega_1 n} \\ &= \cos(\omega_1 n) + \sin(\omega_2 n) + \cos(\omega_3 n) = \cos(\pi n/4) + \sin(\pi n/2) + \cos(3\pi n/4) \end{aligned}$$

which agrees with the above IFFT values at $n = 0, 1, \ldots, 7$. We used the values of the DFT frequencies:

$$\omega_1 = \frac{2\pi}{8} = \frac{\pi}{4}, \quad \omega_2 = \frac{2\pi 2}{8} = \frac{\pi}{2}, \quad \omega_3 = \frac{2\pi 3}{8} = \frac{3\pi}{4}$$

The DFT frequencies ω_7, ω_6, ω_5 are the negatives (modulo-2π) of ω_1, ω_2, ω_3, respectively.

Problem 9.15

Figure P9.3 shows the procedure. The $(4N)$-dimensional time data vector is shuffled once into two $(2N)$-dimensional subvectors by putting every other one of its entries on top and the remaining entries on the bottom. Then, each of the $(2N)$-dimensional vectors is shuffled again into two N-dimensional subvectors.

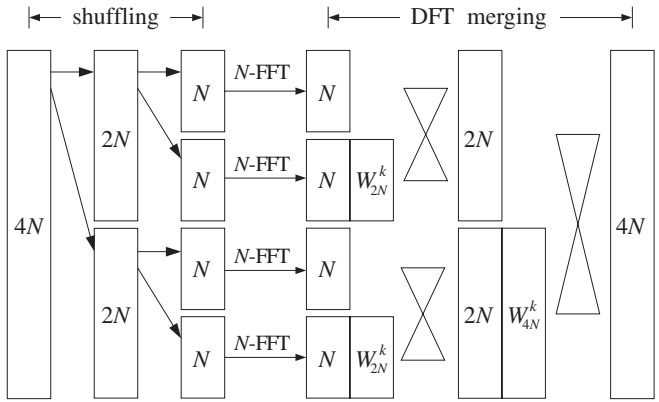


Fig. P9.3 Performing large FFTs.

The four N-dimensional vectors are then shipped over to the FFT processor, one at a time, and their FFTs are returned into four N-dimensional DFT vectors. These are then merged first into $(2N)$-point DFTs and then into the desired $(4N)$-point DFT. The first merging stage require multiplication by the twiddle factors W_{2N}^k, $k = 0, 1, \ldots, N-1$. The second stage uses the factors W_{4N}^k, $k = 0, 1, \ldots, 2N-1$. The butterfly operations can be done is small parts, for example, in groups of N operations at a time so that they can be implemented by the DSP hardware.

The total cost is the cost of four N-point FFTs, the cost of two N-fold multiplications for the first merging stage, and a $(4N)$-fold multiplication for the second stage:

$$\text{cost} = 4\frac{1}{2}N\log_2 N + 2N + 2N = 2N\log_2 N + 4N$$

If the $(4N)$-point FFT could be done in one pass, we would have cost:

$$\text{cost} = \frac{1}{2}(4N)\log_2(4N) = 2N\log_2(4N)$$

It is easily verified that the costs of the two methods are the same. The indirect method is of course much slower because of all the I/O time for shipping data back and forth from secondary storage to the FFT processor.

Problem 9.16

The ordinary convolution is

$$\mathbf{y} = \mathbf{h} * \mathbf{x} = [1, 3, 4, 6, 7, 6, 4, 3, 1]$$

Reducing this mod-4 gives: $\tilde{\mathbf{y}} = [9,9,8,9]$. The alternative method is to reduce first $\mathbf{x}$ and $\mathbf{h}$ mod-4:

$$\tilde{\mathbf{h}} = [2,2,1,2], \qquad \tilde{\mathbf{x}} = [2,1,1,1],$$

then, compute their ordinary convolution:

$$\tilde{\mathbf{h}} * \tilde{\mathbf{x}} = [4,6,6,9,5,3,2],$$

and, finally, reduce that modulo 4 to get $\tilde{\mathbf{y}} = [9,9,8,9]$.

Problem 9.17

Carrying out the 8-point FFT gives:

$$\mathbf{x} = \begin{bmatrix} 4 \\ 2 \\ 4 \\ -6 \\ 4 \\ 2 \\ 4 \\ -6 \end{bmatrix} \quad \xrightarrow{8-FFT} \quad \mathbf{X} = \begin{bmatrix} 8 \\ 0 \\ -16j \\ 0 \\ 24 \\ 0 \\ 16j \\ 0 \end{bmatrix}$$

Taking every other entry of the result gives the 4-point FFT, namely,

$$\mathbf{X} = \begin{bmatrix} 8 \\ -16j \\ 24 \\ 16j \end{bmatrix}$$

And, taking every other entry of that gives the 2-point FFT:

$$\mathbf{X} = \begin{bmatrix} 8 \\ 24 \end{bmatrix}$$

Can we get these results directly? Yes. To do the 4-point FFT, first reduce the signal modulo 4 to get

$$\tilde{\mathbf{x}} = \begin{bmatrix} 8 \\ 4 \\ 8 \\ -12 \end{bmatrix} \quad \xrightarrow{4-FFT} \quad \mathbf{X} = \begin{bmatrix} 8 \\ -16j \\ 24 \\ 16j \end{bmatrix}$$

Putting the above 8-point FFT into the 8-point inverse FFT formula gives

$$x(n) = \frac{1}{8}\sum_{k=0}^{7} X(k)\, e^{j\omega_k n} = \frac{1}{8}\left[8e^{j\omega_0 n} - 16je^{j\omega_2 n} + 24e^{j\omega_4 n} + 16je^{j\omega_6 n}\right]$$

Noting that for an 8-point FFT, $\omega_0 = 0$, $\omega_2 = \pi/2$, $\omega_4 = \pi$, and $\omega_6 = 3\pi/2 \equiv -\pi/2$, and using Euler's formula we obtain:

$$x(n) = 1 + 4\sin\left(\frac{\pi n}{2}\right) + 3\cos(\pi n)$$

which agrees with the given $\mathbf{x}$, for $n = 0, 1, \cdots, 7$.

Problem 9.18

Consider the length-7 signal

$$[1-a,\ 2-b,\ 3,\ 4,\ 5,\ a,\ b]$$

It should be evident that, for any values of the parameters a and b, its mod-5 reduction is $[1,2,3,4,5]$.

Problem 9.19

Using the finite geometric series and the definition $W_N = e^{-2\pi j/N}$, gives

$$\frac{1}{N}\sum_{n=0}^{N-1} W_N^{kn} = \frac{1}{N}\frac{1-W_N^{Nk}}{1-W_N^{k}} = \frac{1}{N}\frac{1-e^{-2\pi jk}}{1-e^{-2\pi jk/N}}$$

If $k \neq 0$, the numerator vanishes but not the denominator. Thus, in this case the answer is zero. If $k = 0$, we have a ratio of $0/0$, but in this case, we can just go back to the series definition and set $k = 0$. All of the N terms in the sum become unity, giving $N/N = 1$.

In conclusion, the answer is zero if $k \neq 0$, and unity if $k = 0$. Note, that these arguments break down if k is allowed to go beyond the range $0 \leq k \leq N-1$, because whenever it is a multiple of N, we would get unity as an answer (and zero otherwise).

Problem 9.20

For a general power p, we have:

$$W_{pN}^{p} = \left(e^{-2\pi j/(pN)}\right)^{p} = e^{-2\pi j/N} = W_N$$

The DTFT of a length-L signal and its N-point and (pN)-point DFTs are:

$$X(\omega) = \sum_{n=0}^{L-1} x(n)\, e^{-j\omega n}$$

$$X_N(k) = \sum_{n=0}^{L-1} x(n)\, W_N^{kn}$$

$$X_{pN}(k) = \sum_{n=0}^{L-1} x(n)\, W_{pN}^{kn}$$

Replacing k by pk in the third and using part (a), we have:

$$X_{pN}(pk) = \sum_{n=0}^{L-1} x(n)\, W_{pN}^{pkn} = \sum_{n=0}^{L-1} x(n)\, W_N^{kn} = X_N(k)$$

Problem 9.21

This follows from the results of Problem 9.20 by setting $N = 8$ and $p = 2$, which implies:

$$X_8(k) = X_{16}(2k), \qquad k = 0, 1, \ldots, 15$$

A geometrical way to understand this is to recognize that the 8th roots of unity are a subset of the 16th roots of unity and are obtained by taking every other one of the latter.

Problem 9.22

Using a trig identity to write the product of sines as a sum of sines, we get the equivalent expression:

$$x(t) = \cos(24\pi t) + 2\sin(12\pi t)\cos(8\pi t) = \cos(24\pi t) + \sin(20\pi t) + \sin(4\pi t)$$

The frequencies of the three terms are 12, 10, and 2 kHz. Their aliased versions that lie inside the Nyquist interval $[-4, 4]$ kHz are, respectively, 4, 2, and 2 kHz. Thus, the aliased signal that would be reconstructed by an ideal reconstructor will be:

$$x(t) = \cos(8\pi t) + \sin(4\pi t) + \sin(4\pi t) = \cos(8\pi t) + 2\sin(4\pi t)$$

Setting $t = nT = n/8$, we get the time samples:

$$x(n) = \cos(\pi n) + 2\sin(\pi n/2)$$

Evaluating it at $n = 0, 1, \ldots, 7$ gives the values, and their FFT:

$$\mathbf{x} = [1, 1, 1, -3, 1, 1, 1, -3] \quad \overset{8-FFT}{\longrightarrow} \quad \mathbf{X} = [0, 0, -8j, 0, 8, 0, 8j, 0]$$

The same DFT values can be determined without any computation by comparing $x(n)$ with the inverse DFT. Using Euler's formula, setting $\omega_4 = \pi$, $\omega_2 = \pi/2$, $\omega_6 = -(\pi/2) \bmod (2\pi)$, and multiplying and dividing by 8, we get:

$$x(n) = e^{j\pi n} - je^{j\pi n/2} + je^{-j\pi n/2} = \frac{1}{8}\left[-8je^{j\omega_2 n} + 8e^{j\omega_4 n} + 8je^{j\omega_6 n}\right]$$

Comparing with the 8-point IDFT, gives the above DFT vector:

$$x(n) = \frac{1}{8}\sum_{k=0}^{7} X(k)\, e^{j\omega_k n}$$

Problem 9.23

Using Euler's formula, we write:

$$\begin{aligned} x(n) &= 1 + 2\sin\left(\frac{\pi n}{4}\right) - 2\sin\left(\frac{\pi n}{2}\right) + 2\sin\left(\frac{3\pi n}{4}\right) + 3(-1)^n \\ &= 1 - je^{j\pi n/4} + je^{-j\pi n/4} + je^{j\pi n/2} - je^{-j\pi n/2} - je^{j3\pi n/4} + je^{-j3\pi n/4} + 3e^{j\pi n} \\ &= \frac{1}{8}\left[8e^{j\omega_0 n} - 8je^{j\omega_1 n} + 8je^{j\omega_2 n} - 8je^{j\omega_3 n} + 24e^{j\omega_4 n} + 8je^{j\omega_5 n} - 8je^{j\omega_6 n} + 8je^{j\omega_7 n}\right] \end{aligned}$$

where we reordered the terms according to increasing DFT index. Comparing with the IDFT formula, we get:

$$\mathbf{X} = [8, -8j, 8j, -8j, 24, 8j, -8j, 8j]$$

Problem 9.24

Using Euler's formula, we write

$$\begin{aligned} x(n) &= 0.5e^{\pi jn/2} + 0.5e^{-\pi jn/2} + e^{\pi jn/8} + e^{-\pi jn/8} \\ &= \frac{1}{16}\left[8e^{\pi jn/2} + 8e^{3\pi jn/2} + 16e^{\pi jn/8} + 16e^{15\pi jn/8}\right] \end{aligned}$$

where, we shifted the negative frequency exponents by adding $2\pi n$ to them, e.g., replaced $-\pi n/8$ by $2\pi n - \pi n/8 = 15\pi n/8$. Now, comparing with the 16-point IDFT formula

$$x(n) = \frac{1}{16}\sum_{k=0}^{15} X(k)\, e^{j\omega_k n}$$

and noting that $\pi/2 = 8\pi/16 = \omega_4$, $3\pi/2 = 24\pi/16 = \omega_{12}$, $\pi/8 = 2\pi/16 = \omega_1$, and $15\pi/8 = 30\pi/16 = \omega_{15}$, we may identify the (non-zero) values of $X(k)$ as follows:

$$X(1) = 16, \quad X(4) = 8, \quad X(12) = 8, \quad X(15) = 16, \quad \text{or,}$$

$$\mathbf{X} = [0,\ 16,\ 0,\ 0,\ 8,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 8,\ 0,\ 0,\ 16]$$

Problem 9.25

Arguing in a similar fashion, we find the (non-zero) values of the 32-point DFT:

$$X(2) = 32, \quad X(8) = 16, \quad X(24) = 16, \quad X(30) = 32$$

Problem 9.26

Write $x(n)$ in its complex sinusoidal form:

$$x(n) = 0.5e^{j\omega_0 n} - je^{j\omega_4 n} + j\omega^{j\omega_{12} n} + 1.5e^{j\omega_8 n} \tag{P9.1}$$

where $\omega_0 = 0$, $\omega_4 = 2\pi 4/16 = \pi/2$, $\omega_{12} = -\omega_4 \bmod (2\pi)$, $\omega_8 = 2\pi 8/16 = \pi$ are the signal frequencies expressed as 16-point DFT frequencies. The DTFT of each complex sinusoid is given by the shifted spectrum of the rectangular window $W(\omega)$ of length-16:

$$X(\omega) = 0.5W(\omega) - jW(\omega - \omega_4) + jW(\omega - \omega_{12}) + 1.5W(\omega - \omega_8)$$

where

$$W(\omega) = \frac{1 - e^{-16j\omega}}{1 - e^{-j\omega}} = \frac{\sin(8\omega)}{\sin(\omega/2)} e^{-15j\omega/2}$$

The magnitude spectrum $|X(\omega)|$ is plotted versus $0 \le \omega \le 2\pi$ in Fig. P9.4. The spectral peaks at the signal frequencies are evident. Dividing the Nyquist interval into 16 equal bins, gives the 16-point DFT.

It so happens that the zeros of the sidelobes of the $W(\omega - \omega_i)$ coincide with 16-point DFT frequencies. Taking every other 16-DFT point gives the 8-point DFT. The peak values are correctly identified by the 16- and 8-point DFTs because the peak frequencies coincide with DFT frequencies.

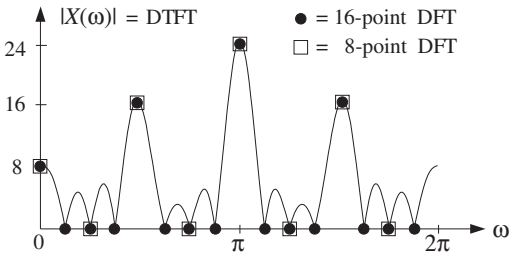


Fig. P9.4 DTFT, 16-point and 8-point DFTs.

The 16-point DFT can be determined from the expression of $x(n)$ by comparing it with 16-point IDFT formula. Multiplying and dividing Eq. (P9.1) by 16, we may extract the DFT coefficients:

$$\mathbf{X} = [8, 0, 0, 0, -16j, 0, 0, 0, 24, 0, 0, 0, 16j, 0, 0, 0]$$

Every other entry is the 8-point DFT:

$$\mathbf{X} = [8, 0, -16j, 0, 24, 0, 16j, 0]$$

Problem 9.27

We will show that the product AA^* is proportional to the identity matrix. The ij-th matrix element of the product is:

$$(AA^*)_{ij} = \sum_{n=0}^{N-1} A_{in} A_{nj}^* = \sum_{n=0}^{N-1} W_N^{in} W_N^{-jn} = \sum_{n=0}^{N-1} W_N^{(i-j)n}$$

where we used the fact that $W_N^* = W_N^{-1}$. Using the results of Problem 9.19, we note that the above sum will be zero if $i \neq j$. And, it will be equal to N if $i = j$. Thus, we can write compactly:

$$(AA^*)_{ij} = \sum_{n=0}^{N-1} W_N^{(i-j)n} = N\delta(i-j)$$

which is equivalent to the matrix relationship

$$AA^* = NI$$

where I is the $N \times N$ identity matrix.

Problem 9.28

Vectorially, the desired equation reads

$$\mathbf{x}^\dagger \mathbf{x} = \frac{1}{N} \mathbf{X}^\dagger \mathbf{X}$$

where † denotes conjugate transpose. Using $\mathbf{X} = A\mathbf{x}$ and the property

$$AA^\dagger = A^\dagger A = NI_N, \qquad (I_N = N \times N \text{ identity matrix})$$

of the $N \times N$ DFT matrix, we obtain

$$\mathbf{X}^\dagger \mathbf{X} = \mathbf{x}^\dagger A^\dagger A \mathbf{x} = \mathbf{x}^\dagger N I_N \mathbf{x} = N \mathbf{x}^\dagger \mathbf{x}$$

Problem 9.29

First, compute the ordinary linear convolution:

$$\mathbf{y} = \mathbf{h} * \mathbf{x} = [2, -1, -2, 2, -2, -1, 2]$$

The mod-4 reduction of this is $\tilde{\mathbf{y}} = [0, -2, 0, 2]$. The mod-5 reduction is $\tilde{\mathbf{y}} = [1, 1, -2, 2, -2]$, the mod-6 reduction is $\tilde{\mathbf{y}} = [4, -1, -2, 2, -2, -1]$. Finally, the mod-7 and mod-8 reductions of $\mathbf{y}$ leave it unchanged. One minor detail is that in the mod-8 case, the length should be $N = 8$, thus, strictly speaking $\tilde{\mathbf{y}}$ is $\mathbf{y}$ with one zero padded at the end, that is, $\tilde{\mathbf{y}} = [2, -1, -2, 2, -2, -1, 2, 0]$. The minimal N is $N = L_y = L_x + L_h - 1 = 4 + 4 - 1 = 7$.

Problem 9.30

First, compute the linear convolution of the given signals:

$$\begin{aligned} \mathbf{y} = \mathbf{h} * \mathbf{x} &= [2,1,1,1,2,1,1,1] * [2,1,2,-3,2,1,2,-3] \\ &= [4,4,7,-1,8,7,11,-2,4,2,1,-1,0,-1,-3] \end{aligned}$$

and wrap it mod-8 to get

$$\tilde{\mathbf{y}} = [8, 6, 8, -2, 8, 6, 8, -2]$$

Then, compute the FFTs of the two signals:

$$\begin{bmatrix} 2 \\ 1 \\ 1 \\ 1 \\ 2 \\ 1 \\ 1 \\ 1 \end{bmatrix} \xrightarrow{8-FFT} \begin{bmatrix} 10 \\ 0 \\ 2 \\ 0 \\ 2 \\ 0 \\ 2 \\ 0 \end{bmatrix}, \qquad \begin{bmatrix} 2 \\ 1 \\ 2 \\ -3 \\ 2 \\ 1 \\ 2 \\ -3 \end{bmatrix} \xrightarrow{8-FFT} \begin{bmatrix} 4 \\ 0 \\ -8j \\ 0 \\ 12 \\ 0 \\ 8j \\ 0 \end{bmatrix}$$

Multiply them point-wise:

$$\begin{bmatrix} 10 \\ 0 \\ 2 \\ 0 \\ 2 \\ 0 \\ 2 \\ 0 \end{bmatrix} \times \begin{bmatrix} 4 \\ 0 \\ -8j \\ 0 \\ 12 \\ 0 \\ 8j \\ 0 \end{bmatrix} = \begin{bmatrix} 40 \\ 0 \\ -16j \\ 0 \\ 24 \\ 0 \\ 16j \\ 0 \end{bmatrix}$$

Divide by $N = 8$, conjugate every entry, and take an 8-point FFT:

$$\begin{bmatrix} 5 \\ 0 \\ 2j \\ 0 \\ 3 \\ 0 \\ -2j \\ 0 \end{bmatrix} \xrightarrow{8-FFT} \begin{bmatrix} 8 \\ 6 \\ 8 \\ -2 \\ 8 \\ 6 \\ 8 \\ -2 \end{bmatrix}$$

Problem 9.31

a. The 8-FFT algorithm gives

$$\mathbf{x} = [6,1,0,1,6,1,0,1] \qquad \Rightarrow \qquad \mathbf{X} = [16,0,12,0,8,0,12,0]$$

b. Using $x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j\omega_k n}$,

$$x(n) = \frac{1}{8}\left[16 + 12e^{j\pi n/2} + 8e^{j\pi n} + 12e^{-j\pi n/2}\right] = 2 + 3\cos(\frac{\pi n}{2}) + (-1)^n$$

c. With arbitrary values of a and b:

$$\mathbf{x} = [6-a,1,0,1,6,1,0,1,a], \qquad \mathbf{x} = [6-a,1-b,0,1,6,1,0,1,a,b]$$

d. Reduce $\mathbf{x}$ mod-4 to get $\tilde{\mathbf{x}} = [12,2,0,2]$. Its 4-FFT is easily found to be $\mathbf{X} = [16,12,8,12]$. It can also be obtained by picking every other entry of the 8-point DFT.

Problem 9.32

Start with

$$A(k) = \sum_{n=0}^{N-1} W_N^{kn} a(n)$$

and evaluate it at $N-k$

$$A(N-k) = \sum_{n=0}^{N-1} W_N^{(N-k)n} a(n)$$

Noting that $W_N^N = 1$, it can be written as

$$A(N-k) = \sum_{n=0}^{N-1} W_N^{-kn} a(n)$$

On the other hand, because $a(n)$ is real

$$A(k)^* = \sum_{n=0}^{N-1} (W_N^{kn})^* a(n)$$

But,

$$(W_N^{kn})^* = W_N^{-kn}$$

Thus,

$$A(N-k) = A(k)^*$$

Problem 9.33

Using linearity, we have

$$X(k) = A(k) + jB(k) \qquad \Rightarrow \qquad X(N-k)^* = A(N-k)^* - jB(N-k)^*$$

Using the results of Problem 9.32, $A(N-k)^* = A(k)$ and $B(N-k)^* = B(k)$, we obtain $X(N-k)^* = A(k) - jB(k)$. Thus, we have the system of equations:

$$X(k) = A(k) + jB(k)$$
$$X(N-k)^* = A(k) - jB(k)$$

which can be solved for $A(k)$ and $B(k)$.

Problem 9.34

The expressions for $G(k)$ and $H(k)$ follow from the results of Problem 9.33, except that the dimension of the FFTs is $N/2$. The final expressions that construct $X(k)$ are the standard Butterfly operations for merging $G(k)$ and $H(k)$.
The additional multiplications to build $X(k)$ from $Y(k)$ are $3N/2$, (that is, the divisions by 2 and $2j$, and multiplications by W_N^k). Thus the total number of multiplications using this method compared to a direct N-point FFT will be

$$\frac{\frac{1}{2}\frac{N}{2}\log_2(\frac{N}{2}) + \frac{3N}{2}}{\frac{1}{2}N\log_2 N} = \frac{1}{2} + \frac{2.5}{\log_2 N}$$

If one takes into account the multiplications required for computing the W^k powers, then the total becomes

$$\frac{\frac{N}{2}\log_2(\frac{N}{2}) + \frac{4N}{2}}{N\log_2 N} = \frac{1}{2} + \frac{1.5}{\log_2 N}$$

Typically, 65% of the standard cost for $N = 1024$.

Problem 9.35

```
/* fftreal.c - FFT of a real-valued signal */

#include <math.h>
#include <cmplx.h>
#include <stdlib.h>

void fft();

void fftreal(N, x, X)
int N;
double *x;                                          time data
complex *X;                                         FFT array
{
       int n, k, Nhalf=N/2;
       double pi=4*atan(1.0);
```

```
        complex one, half, halfj, W;
        complex *Y, temp, Wk, Gk, Hk;

        one = cmplx(1.0, 0.0);
        half = cmplx(0.5, 0.0);
        halfj = cmplx(0.0, -0.5);

        W = cexp(cmplx(0.0, -2*pi/N));

        Y = (complex *) calloc(Nhalf, sizeof(complex));

        for (n=0; n<Nhalf; n++)
               Y[n] = cmplx(x[2*n], x[2*n+1]);

        fft(Nhalf, Y);

        for (Wk=one, k=0; k<Nhalf; k++) {          Wk = W^k
               if (k==0)
                      temp = conjg(Y[0]);          note: Y(0) = Y(N/2)
               else
                      temp = conjg(Y[Nhalf-k]);
               Gk = cmul(half,  cadd(Y[k], temp));   G(k)
               Hk = cmul(halfj, csub(Y[k], temp));   H(k)
               temp = cmul(Wk, Hk);                  W^k H(k)
               Wk = cmul(W, Wk);                     next Wk = W^(k+1)
               X[k] = cadd(Gk, temp);                G(k) + W^k H(k)
               X[k+Nhalf] = csub(Gk, temp);          G(k) - W^k H(k)
               }
}
```

Problem 9.36

An example will demonstrate the result. For $N = 4$, we have:

$$\mathbf{x} = [x_0, x_1, x_2, x_3] \quad \Rightarrow \quad X(z) = x_0 + x_1 z^{-1} + x_2 z^{-2} + x_3 z^{-3}$$

$$\mathbf{x}_R = [x_3, x_2, x_1, x_0] \quad \Rightarrow \quad X_R(z) = x_3 + x_2 z^{-1} + x_1 z^{-2} + x_0 z^{-3}$$

It follows that

$$X_R(z) = z^{-3}[x_3 z^3 + x_2 z^2 + x_1 z + x_0] = z^{-3} X(z^{-1})$$

In the time domain, the operation of reflection leads to the z-transform pairs:

$$x(n) \xrightarrow{z} X(z) \qquad \Rightarrow \qquad x(-n) \xrightarrow{z} X(z^{-1})$$

The operation of delay on the reflected signal then gives:

$$x_R(n) = x(N-1-n) = x(-(n-N+1)) \xrightarrow{z} z^{-(N-1)} X(z^{-1})$$

The N-point DFT of the reversed signal is obtained by setting $z = z_k = e^{2\pi jk/N} = W_N^{-k}$. Using the property $W_N^N = 1$, we find:

$$X_R(k) = z_k^{-N+1} X(z_k^{-1}) = W_N^{(N-1)k} X(-k) = W_N^{-k} X(N-k)$$

where in the last step, we used $X(z_k^{-1}) = X(-\omega_k) = X(-k) = X(N-k)$ by the periodicity of the N-point DFT.

Problem 9.37

In the definition of $\mathbf{y}$, the reversed signal $\mathbf{x}_R$ is delayed by N samples, that is, we have

$$\mathbf{x} = [\mathbf{x}, \mathbf{x}_R] = [\mathbf{x}, \underbrace{0, 0, \ldots, 0}_{N \text{ zeros}}] + [\underbrace{0, 0, \ldots, 0}_{N \text{ zeros}}, \mathbf{x}_R]$$

Thus, using linear superposition, the delay property of z-transforms, and the results of Problem 9.36, we have in the z-domain:

$$Y(z) = X(z) + z^{-N} X_R(z) = X(z) + z^{-2N+1} X(z^{-1})$$

where the factor z^{-N} represents the delay of $\mathbf{x}_R$ by N samples. Replacing z by $z_k = e^{j\omega_k}$, and using the result $z_k^{2N} = 1$, we get

$$\begin{aligned} Y_k = Y(\omega_k) &= X(\omega_k) + e^{j\omega_k} X(-\omega_k) = e^{j\omega_k/2}\left[e^{-j\omega_k/2} X(\omega_k) + e^{j\omega_k/2} X(-\omega_k)\right] \\ &= 2e^{j\omega_k/2} \mathrm{Re}\left[e^{-j\omega_k/2} X(\omega_k)\right] \end{aligned}$$

where we used the hermitian property $X(-\omega) = X(\omega)^*$. The quantity under the real part is now:

$$e^{-j\omega_k/2} X(\omega_k) = e^{-j\omega_k/2} \sum_{n=0}^{N-1} x_n e^{-j\omega_k n} = \sum_{n=0}^{N-1} x_n e^{-j\omega_k (n+1/2)}$$

Thus, its real part is the cosine transform:

$$\mathrm{Re}\left[e^{-j\omega_k/2} X(\omega_k)\right] = \sum_{n=0}^{N-1} x_n \cos\left(\omega_k (n+1/2)\right) = C_k$$

It follows that

$$Y_k = 2e^{j\omega_k/2} C_k = 2e^{j\pi k/2N} C_k$$

Because Y_k is the $(2N)$-point DFT of a real-valued signal, it will satisfy the hermitian property $Y_{2N-k} = Y_k^*$. In terms of C_k, we have:

$$\begin{aligned} Y_{2N-k} &= 2e^{j\pi(2N-k)/2N} C_{2N-k} = 2e^{j\pi} e^{-j\pi k/2N} C_{2N-k} = -2e^{-j\pi k/2N} C_{2N-k} \\ Y_k^* &= 2e^{-j\pi k/2N} C_k \end{aligned}$$

Equating the right-hand sides and canceling the common factors gives the symmetry property

$$C_{2N-k} = -C_k, \qquad k = 0, 1, \ldots, 2N-1$$

At $k = N$, it gives $C_{2N-N} = -C_N$ or $C_N = -C_N$ which implies $C_N = 0$. Next, considering the inverse DFT for Y_k, we have:

$$y_n = \frac{1}{2N} \sum_{k=0}^{2N-1} Y_k e^{j\omega_k n}, \qquad n = 0, 1, \ldots, 2N-1$$

But the first N samples of y_n are precisely x_n. Replacing Y_k in terms of C_k gives for $n = 0, 1, \ldots, N-1$:

$$x_n = \frac{1}{2N} \sum_{k=0}^{2N-1} 2e^{j\omega_k/2} C_k e^{j\omega_k n} = \frac{1}{N} \sum_{k=0}^{2N-1} C_k e^{j\omega_k (n+1/2)}$$

Next, we split the summation into three terms:

$$x_n = \frac{1}{N}\left[C_0 + \sum_{k=1}^{N-1} C_k e^{j\omega_k(n+1/2)} + \sum_{k=N+1}^{2N-1} C_k e^{j\omega_k(n+1/2)}\right]$$

where the second summation starts at $k = N+1$ because $C_N = 0$. If we change variables from k to $m = 2N-k$, we can write

$$\sum_{k=N+1}^{2N-1} C_k e^{j\omega_k(n+1/2)} = \sum_{m=1}^{N-1} C_{2N-m} e^{j\omega_{2N-m}(n+1/2)}$$

Using the symmetry property $C_{2N-m} = -C_m$, and the trig identity

$$e^{j\omega_{2N-m}(n+1/2)} = e^{j\pi(2N-m)(n+1/2)/N} = e^{2j\pi(n+1/2)} e^{-j\pi m(n+1/2)/N} = -e^{-j\omega_m(n+1/2)}$$

we have (with the two minus signs canceling):

$$\sum_{m=1}^{N-1} C_{2N-m} e^{j\omega_{2N-m}(n+1/2)} = \sum_{m=1}^{N-1} C_m e^{-j\omega_m(n+1/2)}$$

Thus, the inverse DFT becomes (changing the summation from m back to k):

$$x_n = \frac{1}{N}\left[C_0 + \sum_{k=1}^{N-1} C_k e^{j\omega_k(n+1/2)} + \sum_{k=1}^{N-1} C_k e^{-j\omega_k(n+1/2)}\right]$$

The two summations now combine with Euler's formula to give:

$$x_n = \frac{1}{N}\left[C_0 + 2\sum_{k=1}^{N-1} C_k \cos\left(\omega_k(n+1/2)\right)\right]$$

Problem 9.38

By definition, we have

$$X_{2N}(k) = \sum_{n=0}^{L-1} x(n) W_{2N}^{nk} \qquad \Rightarrow \qquad X_{2N}(2k) = \sum_{n=0}^{L-1} x(n) W_{2N}^{2nk}$$

But, $W_{2N}^{2nk} = \exp(-2\pi j 2nk/2N) = \exp(-2\pi jnk/N) = W_N^{nk}$. Thus,

$$X_{2N}(2k) = \sum_{n=0}^{L-1} x(n) W_{2N}^{2nk} = \sum_{n=0}^{L-1} x(n) W_N^{nk} = X_N(k)$$

Problem 9.39

Observe that the five length-5 sequences are the mod-5 wrapped versions of the following "linearly shifted" sequences:

$$\begin{array}{cccccccccc}
x_0 & x_1 & x_2 & x_3 & x_4 & 0 & 0 & 0 & 0 & 0 \\
0 & x_0 & x_1 & x_2 & x_3 & x_4 & 0 & 0 & 0 & 0 \\
0 & 0 & x_0 & x_1 & x_2 & x_3 & x_4 & 0 & 0 & 0 \\
0 & 0 & 0 & x_0 & x_1 & x_2 & x_3 & x_4 & 0 & 0 \\
0 & 0 & 0 & 0 & x_0 & x_1 & x_2 & x_3 & x_4 & 0
\end{array}$$

Their z-transforms differ by a simple delay factor, that is,

$$X_i(z) = z^{-i} X_0(z), \qquad i = 1,2,3,4$$

Setting $z = z_k = e^{j\omega_k} = W_5^{-k}$, we obtain

$$X_i(z_k) = z_k^{-i} X_0(z_k) = W_5^{ik} X_0(z_k), \qquad i = 1,2,3,4$$

Problem 9.40

Consider, for example, the first and the third. Their z-transforms are

$$X_1(z) = x_0 + x_1 z^{-1} + x_2 z^{-2} + x_3 z^{-3} + x_4 z^{-4} + x_5 z^{-5} + x_6 z^{-6} + x_7 z^{-7}$$

$$X_3(z) = x_0 + x_1 z^{-1} + (x_2 + x_6) z^{-2} + (x_3 + x_7) z^{-3} + x_4 z^{-4} + x_5 z^{-5}$$

Subtracting, we have

$$X_3(z) - X_1(z) = x_6 z^{-2}(1 - z^{-4}) + x_7 z^{-3}(1 - z^{-4}) = (x_6 z^{-2} + x_7 z^{-3})(1 - z^{-4})$$

Evaluating at the 4th roots of unity, $z = z_k = e^{2\pi jk/4}$, we obtain

$$X_3(z_k) - X_1(z_k) = 0, \qquad k = 0,1,2,3$$

Problem 9.41

Using the convolution table, perform the ordinary linear convolution of the given signal and filter to get

$$\mathbf{y} = \mathbf{h} * \mathbf{x} = [1,0,-1,0,2,0,-2,0,-2,0,2,1,0,-1,0,-1,0,1,0,-1,0,1]$$

The overlap-save method divides the input signal into segments of length $N = 8$ that are overlapping by $M = 3$ points, as shown below:

$$\mathbf{x} = \Big(1,1,1,1,3,(3,3,3)\Big), 1, 1, \Big(1,2,2),2,2,(1,1,1\Big), 1,0,0,0,0)$$

Convolving these segments with the filter gives:

$$\mathbf{y}_0 = [1,-1,-1,1] * [1,1,1,1,3,3,3,3] = [1,0,-1,0,2,0,-2,0,-3,0,3]$$

$$\mathbf{y}_1 = [1,-1,-1,1] * [3,3,3,1,1,1,2,2] = [3,0,-3,-2,0,2,1,0,-3,0,2]$$

$$\mathbf{y}_2 = [1,-1,-1,1] * [1,2,2,2,2,1,1,1] = [1,1,-1,-1,0,-1,0,1,-1,0,1]$$

$$\mathbf{y}_3 = [1,-1,-1,1] * [1,1,1,1,0,0,0,0] = [1,0,-1,0,-1,0,1,0,0,0,0]$$

Reducing these mod-8 and ignoring their first $M = 3$ points (indicated by x), we obtain:

$$\tilde{\mathbf{y}}_0 = [x,x,x,0,2,0,-2,0]$$

$$\tilde{\mathbf{y}}_1 = [x,x,x,-2,0,2,1,0]$$

$$\tilde{\mathbf{y}}_2 = [x,x,x,-1,0,-1,0,1]$$

$$\tilde{\mathbf{y}}_3 = [x,x,x,0,-1,0,1,0]$$

Abutting the rest of the points, we get

$$\mathbf{y} = [x,x,x,0,2,0,-2,0][-2,0,2,1,0][-1,0,-1,0,1][0,-1,0,1,0]$$

With the exception of the first M points, the answer is correct.

Problem 9.42

At $f_s = 8$ Hz, there are 8 samples per period. These samples can be evaluated by setting $t = nT = n/8$ in the expression for $x(t)$. We find:

$$x(n) = \frac{1}{4}\{0, 0.5, 1, 0.5, 0, -0.5, -1, -0.5\}$$

According to Eq. (9.7.4), the aliased coefficients can be obtained by computing the 8-point DFT of one period and dividing by 8. This DFT can be done quickly by an 8-point FFT by hand, which gives:

$$\mathbf{x} = \frac{1}{4}\begin{bmatrix} 0 \\ 0.5 \\ 1 \\ 0.5 \\ 0 \\ -0.5 \\ -1 \\ -0.5 \end{bmatrix} \quad \overset{\text{DFT}}{\longrightarrow} \quad \mathbf{X} = \frac{1}{4}\begin{bmatrix} 0 \\ -j(2+\sqrt{2}) \\ 0 \\ j(2-\sqrt{2}) \\ 0 \\ -j(2-\sqrt{2}) \\ 0 \\ j(2+\sqrt{2}) \end{bmatrix}, \quad \mathbf{b} = \frac{1}{4}\begin{bmatrix} 0 \\ (2+\sqrt{2})/8j \\ 0 \\ -(2-\sqrt{2})/8j \\ 0 \\ (2-\sqrt{2})/8j \\ 0 \\ -(2+\sqrt{2})/8j \end{bmatrix}$$

The coefficients b_1 and b_7 correspond to frequencies $\pm f_1 = \pm 1$ Hz, whereas b_3 and b_5 correspond to $\pm f_2 = \pm 3$ Hz. Thus, we find from Eq. (9.7.5) and after using Euler's formula:

$$x_{\text{al}}(t) = 2jb_1 \sin(2\pi f_1 t) + 2jb_3 \sin(2\pi f_2 t) = \frac{2+\sqrt{2}}{16}\sin(2\pi t) - \frac{2-\sqrt{2}}{16}\sin(6\pi t)$$

It can be verified easily that x_{al} and $x(t)$ agree at the sampling instants $t = nT = n/8$.

Problem 9.43

In the overlap-add method, the input blocks have length $N - M$ and are non-overlapping. Thus, if there are K blocks in the total length, we will have:

$$L = (N-M)K \tag{P9.2}$$

In Fig. 9.9.1, the input blocks overlap by M points, but again if there are K segments, the total length will be $(N-M)K$ plus an extra M points at the end. Thus, we have in the overlap-save method:

$$L = (N-M)K + M \simeq (N-M)K$$

which is essentially the same as in Eq. (P9.2) for large L. Thus, in both cases, the total number of segments can be estimated to be:

$$K = \frac{L}{N-M}$$

Per segment, we are doing two FFTs and N multiplications, namely,

$$N\log_2 N + N = N(1+\log_2 N) \quad \text{per segment}$$

Thus, for K segments, we have cost:

$$KN(1+\log_2 N) = \frac{LN(1+\log_2 N)}{N-M}$$

This is to be compared with the cost of $L(M+1)$ of linear convolution of the entire input array with the filter (here, we treat these MACs as complex-valued.) Thus, the relative cost of the fast versus slow convolution will be:

$$\frac{\text{fast}}{\text{slow}} = \frac{\dfrac{LN(1+\log_2 N)}{N-M}}{L(M+1)} = \frac{N(1+\log_2 N)}{(N-M)(M+1)}$$

Problem 9.44

An example of such a program is given below:

```
/* ovsave.c - FIR filtering by overlap-save method and FFT */

#include <stdio.h>
#include <cmplx.h>
#include <stdlib.h>                                   calloc(), realloc()

#define MAX 64                                        initial allocation size

void circonv(), complexify(), fft(), ifft();

void main(int argc, char **argv)
{
    FILE *fph;                                        filter file
    double *h, *x, *xhead;                            x = length-N block
    complex *H, *X, *Y;                               N-point DFTs
    int M, N, n, i;
    int max = MAX, dmax = MAX;

    if (argc != 3) {
        fprintf(stderr, "usage: fastconv hfile N < xfile  > yfile\n");
        exit(0);
        }

    if ((fph = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "can't open filter file: %s\n", argv[1]);
        exit(0);
        }

    N = atoi(argv[2]);

    h = (double *) calloc(max + 1, sizeof(double));   preliminary allocation

    for (M=0;; M++) {                                 read h
        if (M == max) {                               reallocate h, if necessary
               max += dmax;
               h = (double *) realloc((char *) h, (max + 1) * sizeof(double));
               }
        if (fscanf(fph, "%lf", h + M) == EOF) break;
        }

    M--;                                              M = filter order

    if (N <= 2*M) {
        fprintf(stderr, "must have N > 2M = %d", 2*M);
        exit(0);
        }
```

```
    h     = (double *) realloc((char *) h, (M+1) * sizeof(double));
    x     = (double *) calloc(N, sizeof(double));
    xhead = (double *) calloc(M, sizeof(double));        head/tail of input block

    H = (complex *) calloc(N, sizeof(complex));          allocate DFT arrays
    X = (complex *) calloc(N, sizeof(complex));
    Y = (complex *) calloc(N, sizeof(complex));

    for (n=0; n<N; n++)                                  complexify/extend to length N
        if (n <= M)
               H[n] = cmplx(h[n], 0.0);
        else
               H[n] = cmplx(0.0, 0.0);

    fft(N, H);                                           H = FFT(h), done once

    for (n=0; n<M; n++)
        xhead[n] = 0;                                    pad M zeros in front of first block

    for (;;) {                                           keep reading input in blocks x
        for (n=0; n<M; n++)                              previous tail is head of present block x
            x[n] = xhead[n];

        for (n=M; n<N; n++)                              read rest of x from input stream
            if (scanf("%lf", x + n) == EOF) goto lastblocks;

        for (n=0; n<M; n++)                              tail of present block x
            xhead[n] = x[n + N - M];                     will be head of next block

        complexify(N, x, X);                             X = cmplx(x, 0)

        circonv(N, H, X, Y);                             done by fft

        for (n=M; n<N; n++)                              discard first M points, save the rest
               printf("%lf\n", real(Y[n]));
        }

lastblocks:
                                                         n = index where EOF occurred
        for (i=n; i<N; i++)                              pad rest of x with zeros
            x[i] = 0;                                    if n <= N-M there is only
                                                         one last block to consider
        complexify(N, x, X);                             otherwise, there are two last blocks

        circonv(N, H, X, Y);

        for (i=M; i<N; i++)
            printf("%lf\n", real(Y[i]));

        if (n <= N-M) exit(0);                           exit, if only one last block
                                                         otherwise, process very last block
        for (i=0; i<M; i++)                              if n > N-M, the tail of previous
            xhead[i] = x[i + N - M];                     block will be non-zero

        for (i=0; i<N; i++)
            if (i < M)
                x[i] = xhead[i];                         head of very last block
```



```
            else
                x[i] = 0;                                rest of block is zero

        complexify(N, x, X);

        circonv(N, H, X, Y);

        for (i=M; i<N; i++)
            printf("%lf\n", real(Y[i]));
}

--------------------------

void complexify(N, x, X)
int N;
double *x;
complex *X;
{
    int n;

    for (n=0; n<N; n++)
        X[n] = cmplx(x[n], 0.0);
}

--------------------------

void circonv(N, H, X, Y)                                 circular convolution via FFT
int N;
complex *H, *X, *Y;                                      H = FFT(h), must be done previously
{
    int k;

    fft(N, X);

    for (k=0; k<N; k++)
        Y[k] = cmul(H[k], X[k]);

    ifft(N, Y);
}
```

Chapter 10 Problems

Problem 10.1

Start with

$$D(\omega) = \sum_{k=-\infty}^{\infty} d(k)\, e^{-j\omega k}$$

and assume that $d(k)$ is real and even in k. Taking conjugates and replacing ω by $-\omega$, gives

$$D(-\omega)^* = \Big[\sum_{k=-\infty}^{\infty} d(k)\, e^{j\omega k}\Big]^* = \sum_{k=-\infty}^{\infty} d(k)\, e^{-j\omega k} = D(\omega)$$

This result uses only the reality of $d(k)$ and is the usual hermitian property for transfer functions of real-valued impulse responses. Because $d(k) = d(-k)$, we can change the summation from k to $-k$ to get:

$$D(\omega) = \sum_{k=-\infty}^{\infty} d(k)\, e^{-j\omega k} = \sum_{k=-\infty}^{\infty} d(-k)\, e^{-j\omega(-k)} = \sum_{k=-\infty}^{\infty} d(k)\, e^{-j(-\omega)k} = D(-\omega)$$

Thus, the reality and evenness conditions imply together:

$$D(\omega) = D(-\omega)^* = D(-\omega)$$

which states that $D(\omega)$ is both real and even in ω. In the antisymmetric case, we have $d(-k) = -d(k)$, and therefore

$$D(\omega) = \sum_{k=-\infty}^{\infty} d(k)\, e^{-j\omega k} = \sum_{k=-\infty}^{\infty} d(-k)\, e^{-j\omega(-k)} = -\sum_{k=-\infty}^{\infty} d(k)\, e^{-j(-\omega)k} = -D(-\omega)$$

Together with the reality condition, we get

$$D(\omega) = D(-\omega)^* = -D(-\omega)$$

which implies that $D(\omega)$ is pure imaginary and odd in ω. Separating the negative- and positive-k terms, we can rewrite $D(\omega)$ in the form:

$$D(\omega) = d(0) + \sum_{k=1}^{\infty} d(k)\, e^{-j\omega k} + \sum_{k=-\infty}^{-1} d(k)\, e^{-j\omega k} = d(0) + \sum_{k=1}^{\infty} d(k)\, e^{-j\omega k} + \sum_{k=1}^{\infty} d(-k)\, e^{j\omega k}$$

In the symmetric case, we set $d(-k) = d(k)$ and combine the two sums with Euler's formula to get:

$$D(\omega) = d(0) + 2\sum_{k=1}^{\infty} d(k)\cos(\omega k) \qquad \text{(symmetric case)}$$

For the antisymmetric case, $d(0) = 0$ because it must satisfy $d(-0) = -d(0)$. Setting $d(-k) = -d(k)$ in the sum, we obtain:

$$D(\omega) = -2j\sum_{k=1}^{\infty} d(k)\sin(\omega k) \qquad \text{(antisymmetric case)}$$

Problem 10.2

For the bandpass example, we use the fact that $D(\omega) = 1$ over the passband $\omega_a \le |\omega| \le \omega_b$ to get:

$$\begin{aligned} d(k) &= \int_{-\pi}^{\pi} D(\omega)\, e^{j\omega k}\, \frac{d\omega}{2\pi} = \int_{-\omega_b}^{-\omega_a} D(\omega)\, e^{j\omega k}\, \frac{d\omega}{2\pi} + \int_{\omega_a}^{\omega_b} D(\omega)\, e^{j\omega k}\, \frac{d\omega}{2\pi} = \\ &= \int_{-\omega_b}^{-\omega_a} e^{j\omega k}\, \frac{d\omega}{2\pi} + \int_{\omega_a}^{\omega_b} e^{j\omega k}\, \frac{d\omega}{2\pi} = \left[\frac{e^{j\omega k}}{2\pi jk}\right]_{-\omega_b}^{-\omega_a} + \left[\frac{e^{j\omega k}}{2\pi jk}\right]_{\omega_a}^{\omega_b} \\ &= \frac{e^{-j\omega_a k} - e^{-j\omega_b k}}{2\pi jk} + \frac{e^{j\omega_b k} - e^{j\omega_a k}}{2\pi jk} = \frac{\sin(\omega_b k) - \sin(\omega_a k)}{\pi k} \end{aligned}$$

Setting $\omega_a = 0$, gives the lowpass case, and $\omega_b = \pi$ the highpass one. In the latter, we have:

$$d(k) = \frac{\sin(\pi k) - \sin(\omega_b k)}{\pi k}$$

The first term is zero for all non-zero k, and its equal to unity for $k = 0$. Thus, we may replace it by $\delta(k)$:

$$d(k) = \delta(k) - \frac{\sin(\omega_a k)}{\pi k}$$

For the differentiator and Hilbert transformer, see the more general cases of Problem 10.3.

Problem 10.3

For the lowpass differentiator, we have

$$d(k) = \int_{-\pi}^{\pi} D(\omega)\, e^{j\omega k}\, \frac{d\omega}{2\pi} = \int_{-\omega_c}^{\omega_c} j\omega\, e^{j\omega k}\, \frac{d\omega}{2\pi}$$

It can be evaluated either by parts, or by recognizing the integrand as the derivative with respect to k (treating k initially as a real variable and at the end restricting it to integer values.)

$$d(k) = \frac{\partial}{\partial k}\int_{-\omega_c}^{\omega_c} e^{j\omega k}\, \frac{d\omega}{2\pi} = \frac{\partial}{\partial k}\left[\frac{\sin(\omega_c k)}{\pi k}\right]$$

where the ω-integration gave the standard lowpass filter. Performing the differentiation, we get the required expression for $d(k)$:

$$d(k) = \frac{\omega_c \cos(\omega_c k)}{\pi k} - \frac{\sin(\omega_c k)}{\pi k^2}$$

Setting $\omega_c = \pi$ gives the full-band differentiator:

$$d(k) = \frac{\cos(\pi k)}{k} - \frac{\sin(\pi k)}{\pi k^2}$$

Because $d(k)$ belongs to the antisymmetric class, it will have $d(0) = 0$. The second term vanishes for nonzero integer ks. Thus, we have for $k \neq 0$:

$$d(k) = \frac{\cos(\pi k)}{k} = \frac{(-1)^k}{k} \qquad \text{(differentiator)}$$

The Hilbert transformer case is similar.

Problem 10.4

Using the differentiation argument, we have in this case:

$$d(k) = \frac{\partial}{\partial k}\left[\int_{-\omega_b}^{-\omega_a} e^{j\omega k}\,\frac{d\omega}{2\pi} + \int_{\omega_a}^{\omega_b} e^{j\omega k}\,\frac{d\omega}{2\pi}\right]$$

The integrations give the bandpass filter case. Thus,

$$d(k) = \frac{\partial}{\partial k}\left[\frac{\sin(\omega_b k) - \sin(\omega_a k)}{\pi k}\right]$$

which becomes the difference of two lowpass differentiators:

$$d(k) = \frac{\omega_b \cos(\omega_b k)}{\pi k} - \frac{\sin(\omega_b k)}{\pi k^2} - \frac{\omega_a \cos(\omega_a k)}{\pi k} + \frac{\sin(\omega_a k)}{\pi k^2}$$

Problem 10.5

For the first-order differentiator, we have impulse response, transfer function, and frequency response:

$$\mathbf{h} = [1, -1] \qquad \Rightarrow \qquad H(z) = 1 - z^{-1} \qquad \Rightarrow \qquad H(\omega) = 1 - e^{-j\omega}$$

Thus, the NRR will be

$$NRR = h_0^2 + h_1^2 = 1 + 1 = 2$$

Thus, the noise power doubles at the output. To see how closely it resembles the ideal differentiator, we expand the frequency response at low frequencies:

$$H(\omega) = 1 - e^{-j\omega} \simeq 1 - (1 - j\omega) = j\omega \qquad \text{for small } \omega$$

For the ideal lowpass differentiator case, the NRR can be computed as:

$$NRR = \int_{-\omega_c}^{\omega_c} |D(\omega)|^2\,\frac{d\omega}{2\pi} = \int_{-\omega_c}^{\omega_c} |j\omega|^2\,\frac{d\omega}{2\pi} = \frac{\omega_c^3}{3\pi}$$

Thus, it increases rapidly with ω_c. Compared to the full-band maximum NRR obtained with $\omega_c = \pi$, we have $NRR_{\max} = \pi^3/3\pi = \pi^2/3$ and therefore,

$$\frac{NRR}{NRR_{\max}} = \left(\frac{\omega_c}{\pi}\right)^3$$

Problem 10.6

Let $E(\omega) = D(\omega) - \hat{D}(\omega)$ be the error in the designed filter. Then, we have

$$E(\omega) = D(\omega) - \hat{D}(\omega) = \sum_{k=-\infty}^{\infty} d(k)\,e^{-j\omega k} - \sum_{k=-M}^{M} d(k)\,e^{-j\omega k} = \sum_{|k|>M} d(k)\,e^{-j\omega k}$$

because the coefficients $d(k)$ cancel for $|k| \le M$. Thus, the time-domain coefficients of the error are:

$$e(k) = \begin{cases} 0 & \text{if } |k| \le M \\ d(k) & \text{if } |k| > M \end{cases}$$

Parseval's identity applied to $e(k)$ gives then

$$\int_{-\pi}^{\pi} |E(\omega)|^2\,\frac{d\omega}{2\pi} = \sum_{k=-\infty}^{\infty} e^2(k) = \sum_{|k|>M} d^2(k)$$

or,

$$\mathcal{E}_M = \int_{-\pi}^{\pi} |D(\omega) - \hat{D}(\omega)|^2\,\frac{d\omega}{2\pi} = \sum_{|k|>M} d^2(k)$$

The right-hand side sum can be written with the help of Parseval's identity applied to $D(\omega)$ itself:

$$\sum_{|k|>M} d^2(k) = \sum_{k=-\infty}^{\infty} d^2(k) - \sum_{|k|\le M} d^2(k) = \int_{-\pi}^{\pi} |D(\omega)|^2\,\frac{d\omega}{2\pi} - \sum_{k=-M}^{M} d(k)^2$$

In the limit as $M \to \infty$, the second term in the right-hand side tends to the Parseval limit, thus canceling the first term and resulting in $\mathcal{E}_M \to 0$.

Problem 10.7

From Problem 10.3, we have $d(k) = (-1)^k/k$, $k \neq 0$, for the differentiator filter. Using Parseval's identity we have

$$\sum_{k=-\infty}^{\infty} d^2(k) = \int_{-\pi}^{\pi} |D(\omega)|^2\,\frac{d\omega}{2\pi} = \int_{-\pi}^{\pi} \omega^2\,\frac{d\omega}{2\pi} = \frac{\pi^2}{3}$$

But the left-hand side is twice the required sum:

$$\sum_{k=-\infty}^{\infty} d^2(k) = \sum_{k\neq 0} \frac{1}{k^2} = 2\sum_{k=1}^{\infty} \frac{1}{k^2}$$

Thus,

$$2\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{3} \qquad \Rightarrow \qquad \sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$$

For the full-band Hilbert transformer, we have

$$d(k) = \frac{1 - \cos(\pi k)}{\pi k} = \frac{1 - (-1)^k}{\pi k}, \qquad k \neq 0$$

Thus, $d(k)$ is nonzero only for odd k, and is in this case $d(k) = 2/\pi k$. Parseval's identity applied to this filter gives:

$$\sum_{k=-\infty}^{\infty} d^2(k) = \int_{-\pi}^{\pi} |D(\omega)|^2\,\frac{d\omega}{2\pi} = \int_{-\pi}^{\pi} |j\,\mathrm{sign}(\omega)|^2\,\frac{d\omega}{2\pi} = \int_{-\pi}^{\pi} 1\,\frac{d\omega}{2\pi} = 1$$

The left-hand side on the other hand becomes:

$$\sum_{k=-\infty}^{\infty} d^2(k) = 2\sum_{\substack{k=1 \\ k=\text{odd}}}^{\infty} \frac{4}{\pi^2 k^2}$$

Thus, we obtain

$$2\sum_{\substack{k=1\\k=\text{odd}}}^{\infty}\frac{4}{\pi^2k^2}=1\qquad\Rightarrow\qquad\sum_{\substack{k=1\\k=\text{odd}}}^{\infty}\frac{1}{k^2}=\frac{\pi^2}{8}$$

If the sum for all integers k is $\pi^2/6$ and for odd ones $\pi^2/8$, it follows by subtraction that the sum over even integers will be:

$$\sum_{\substack{k=2\\k=\text{even}}}^{\infty}\frac{1}{k^2}=\frac{\pi^2}{6}-\frac{\pi^2}{8}=\frac{\pi^2}{24}$$

Problem 10.8

The sinusoidal response of $D(\omega)$ in its steady-state form is:

$$e^{j\omega n}\xrightarrow{D}D(\omega)e^{j\omega n}=-j\text{sign}(\omega)e^{j\omega n}$$

Replacing ω by $-\omega$, and using the odd symmetry property of $D(\omega)$, we have:

$$e^{-j\omega n}\xrightarrow{D}D(-\omega)e^{-j\omega n}=j\text{sign}(\omega)e^{-j\omega n}$$

But $\cos(\omega n)=[e^{j\omega n}+e^{-j\omega n}]/2$. Thus, forming the same linear combination of outputs, we have:

$$\cos(\omega n)\xrightarrow{D}\frac{1}{2}\left[-j\text{sign}(\omega)e^{j\omega n}+j\text{sign}(\omega)e^{-j\omega n}\right]=\text{sign}(\omega)\sin(\omega n)$$

Similarly, because $\sin(\omega n)=[e^{j\omega n}-e^{-j\omega n}]/2j$, we have for the outputs:

$$\sin(\omega n)\xrightarrow{D}\frac{1}{2j}\left[-j\text{sign}(\omega)e^{j\omega n}-j\text{sign}(\omega)e^{-j\omega n}\right]=-\text{sign}(\omega)\cos(\omega n)$$

Problem 10.9

From the definition of $\hat{D}(z)$, we have:

$$\hat{D}(z)=\sum_{k=-M}^{M}d(k)z^{-k}\qquad\Rightarrow\qquad\hat{D}(z^{-1})=\sum_{k=-M}^{M}d(k)z^{k}=\sum_{k=-M}^{M}d(-k)z^{-k}$$

where in the last step we changed summation from k to $-k$. In the symmetric/antisymmetric cases, we have $d(k)=\pm d(k)$. Therefore,

$$\hat{D}(z^{-1})=\pm\sum_{k=-M}^{M}d(k)z^{-k}=\pm\hat{D}(z)$$

From the reality of $d(k)$, it follows that the zeros will come in conjugate pairs if they are complex. Thus, if z_0 is a complex zero, then z_0^* is also a zero. From the symmetry or antisymmetry properties, it follows that if z_0 is a zero, then

$$\hat{D}(z_0^{-1})=\pm\hat{D}(z_0)=0$$

Thus, $1/z_0$ and $1/z_0^*$ are both zeros. The four zeros $\{z_0,z_0^*,\frac{1}{z_0},\frac{1}{z_0^*}\}$ are shown in Fig. P10.1. If z_0 lies on the unit circle then because $1/z_0^*=z_0$, the only other distinct zero is the complex conjugate.

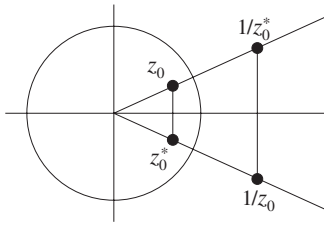


Fig. P10.1 Pattern of zeros of linear phase filters.

In the antisymmetric case, we have in particular for $z=\pm1$,

$$\hat{D}(\pm1)=\sum_{k=-M}^{M}d(k)\,(\pm1)^k=0$$

This follows because $d(k)$ is odd and summed over symmetric limits and because the factor $(\pm1)^k$ is even in k. The same conclusion follows from the frequency response. Using the antisymmetry property and splitting the sum in two parts, we get:

$$\hat{D}(\omega)=\sum_{k=-M}^{M}d(k)e^{-j\omega k}=\sum_{k=1}^{M}d(k)\left[e^{-j\omega k}-e^{j\omega k}\right]=-2j\sum_{k=1}^{M}d(k)\sin(\omega k)$$

Setting $\omega=0,\pi$, corresponding to $z=\pm1$, we have $\sin(\omega k)=0$, and therefore, $\hat{D}(\omega)$ vanishes. Finally, we note that if $d(k)$ is windowed by a window $w(k)$, that is, $d'(k)=w(k)w(k)$, then because the window $w(k)$ is symmetric with respect to the origin, the symmetry/antisymmetry properties of $d(k)$ will be preserved and will imply similar results on the locations of the zeros.

Problem 10.10

For the symmetric case, the following must necessarily be zeros:

$$z_0=0.5+0.5j,\quad z_0^*=0.5-0.5j,\quad\frac{1}{z_0}=1-j,\quad\frac{1}{z_0^*}=1+j$$

Thus, the transfer function will be

$$H(z)=\left(1-(0.5+0.5j)z^{-1}\right)\left(1-(0.5-0.5j)z^{-1}\right)\left(1-(1-j)z^{-1}\right)\left(1-(1+j)z^{-1}\right)=1-3z^{-1}+4.5z^{-2}-3z^{-3}+z^{-4}$$

In the time domain, we get the length-5 symmetric (about its middle) impulse response:

$$\mathbf{h}=[1,\,-3,\,4.5,\,-3,\,1]$$

In the antisymmetric case, we have in addition two zeros at $z=\pm1$ contributing a factor $(1-z^{-2})$ to the transfer function. Thus,

$$H(z)=(1-z^{-2})(1-3z^{-1}+4.5z^{-2}-3z^{-3}+z^{-4})=1-3z^{-1}+3.5z^{-2}-3.5z^{-4}+3z^{-5}-z^{-6}$$

and in the time domain:

$$\mathbf{h}=[1,\,-3,\,3.5,\,0,\,-3.5,\,3,\,-1]$$

The frequency response in the symmetric case is obtained by factoring out a factor of z^{-2} and using Euler's formula:

$$H(z) = z^{-2}\left[z^2 + z^{-2} - 3(z + z^{-1}) + 4.5\right] \qquad \Rightarrow$$

$$H(\omega) = 2e^{-2j\omega}\left[\cos(2\omega) - 3\cos(\omega) + 4.5\right]$$

For the antisymmetric case, we have:

$$H(z) = z^{-3}\left[(z^3 - z^{-3}) - 3(z^2 - z^{-2}) + 3.5(z - z^{-1})\right] \qquad \Rightarrow$$

$$H(\omega) = -2je^{-3j\omega}\left[\sin(3\omega) - 3\sin(2\omega) + 3.5\sin(\omega)\right]$$

Problem 10.11

Proceeding as in the previous problem, we find the transfer function of the symmetric case:

$$H(z) = (1-0.5jz^{-1})(1+0.5jz^{-1})(1-2jz^{-1})(1+2jz^{-1}) = 1+4.25z^{-2}+z^{-4} \qquad \Rightarrow \qquad \mathbf{h} = [1, 0, 4.25, 0, 1]$$

and for the antisymmetric case:

$$H(z) = (1-z^{-2})(1+4.25z^{-2}+z^{-4}) = 1+3.25z^{-2}-3.25z^{-4}-z^{-6} \qquad \Rightarrow \qquad \mathbf{h} = [1,\ 0,\ 3.25,\ 0,\ -3.25,\ 0,\ -1]$$

Problem 10.12

We use the first-order Taylor series approximation:

$$\ln(1 \pm x) \simeq \pm x$$

Changing to base-10, we have:

$$\log_{10}(1 \pm x) = \frac{\ln(1 \pm x)}{\ln(10)} = \pm 0.4343\, x$$

It follows that

$$A_{\text{pass}} = 20\log_{10}\left(\frac{1+\delta_{\text{pass}}}{1-\delta_{\text{pass}}}\right) = 20\log_{10}(1+\delta_{\text{pass}}) - 20\log_{10}(1-\delta_{\text{pass}})$$

$$\simeq 20 \cdot 0.4343\,\delta_{\text{pass}} + 20 \cdot 0.4343\,\delta_{\text{pass}} = 17.372\,\delta_{\text{pass}}$$

Problem 10.13

First, we calculate δ_{pass} and δ_{stop} from Eq. (10.2.5):

$$\delta_{\text{pass}} = \frac{10^{0.1/20} - 1}{10^{0.1/20} + 1} = 0.0058, \qquad \delta_{\text{stop}} = 10^{-80/20} = 0.0001$$

Therefore, $\delta = \min(\delta_{\text{pass}}, \delta_{\text{stop}}) = \delta_{\text{stop}} = 0.0001$, which in dB is $A = -20\log_{10}\delta = A_{\text{stop}} = 80$. The D and α parameters are computed by:

$$\alpha = 0.1102(A - 8.7) = 0.1102(80 - 8.7) = 7.857, \quad D = \frac{A - 7.95}{14.36} = 5.017$$

The transition width is $\Delta f = f_{\text{stop}} - f_{\text{pass}} = 2 - 1.5 = 0.5$ kHz; therefore, the filter length will be:

$$N = 1 + \frac{Df_s}{\Delta f} = 1 + \frac{5.017 \cdot 10}{0.5} = 101.34 \qquad \Rightarrow \qquad N = 103$$

Problem 10.14

The transition width is inversely related to the filter length N:

$$\Delta f = \frac{Df_s}{N-1}$$

As in the previous problem, we find $D = 5.017$. Thus,

$$\Delta f = \frac{Df_s}{N-1} = \frac{5.017 \cdot 10}{251 - 1} = 0.2007 \text{ kHz}$$

Problem 10.15

The window's width factor is $D = 5.017$ as in the previous problem. Thus,

$$\Delta f = \frac{Df_s}{N-1} = \frac{5.017 \cdot 44.1}{129 - 1} = 1.729 \text{ kHz}$$

Conversely, if $\Delta f = 2$ kHz, then we may solve for the D-factor and attenuation A:

$$D = \frac{A - 7.95}{14.36} = (N-1)\frac{\Delta f}{f_s} = 5.805 \qquad \Rightarrow \qquad A = 14.36D + 7.95 = 91.31 \text{ dB}$$

The corresponding ripple will be

$$\delta = 10^{-A/20} = 10^{-91.31/20} = 0.000027$$

which gives the passband attenuation:

$$A_{\text{pass}} = 17.372\,\delta = 0.000472 \text{ dB}$$

In the 4-times oversampling case, the transition width is required to be:

$$\Delta f = 24.55 - 19.55 = 5 \text{ kHz}$$

This gives the width factor and attenuation:

$$D = \frac{A - 7.95}{14.36} = (N-1)\frac{\Delta f}{4f_s} = 3.6281 \qquad \Rightarrow \qquad A = 14.36D + 7.95 = 60.05 \text{ dB}$$

Problem 10.16

Assuming that an N-tap filter requires approximately N instructions for processing each input sample, the maximum filter length will correspond to the maximum number of instructions that fit within the sampling interval T, that is,

$$N_{\max} = \frac{T}{T_{\text{instr}}} = \frac{f_{\text{instr}}}{f_s}$$

Inserting this into the Kaiser design formula, we find the minimum width (where we used N instead of $N-1$ in the denominator):

$$\Delta f = \frac{Df_s}{N} = \frac{Df_s^{\,2}}{f_{\text{instr}}}$$

Problem 10.17

Using $N = f_{\text{instr}}/f_s$ for the maximum filter length, we obtain the maximum width parameter D and attenuation A:

$$D = \frac{A - 7.95}{14.36} = N\frac{\Delta f}{f_s} = \frac{f_{\text{instr}}\Delta f}{{f_s}^2} = F_{\text{instr}}\Delta F \qquad \Rightarrow \qquad A = 14.36F_{\text{instr}}\Delta F + 7.95$$

Problem 10.18

The following MATLAB program segment computes the rectangularly and Hamming windowed impulse responses with the help of the MATLAB function `dlh.m`, and then it computes the corresponding frequency responses over N_F equally spaced frequencies over the right half of the Nyquist interval $[0, \pi]$ with the help of the function `dtft.m`:

```
wc = 0.3 * pi;                                  % cutoff frequency
N = 81;                                         % filter length

w = 0.54 - 0.46 * cos(2 * pi * (0:N-1) / (N-1));   % Hamming window

hrec = dlh(1, wc, N);                           % ideal lowpass filter
hham = w .* hrec;                               % windowed filter

NF = 200;                                       % 200 frequencies in rads/sample
omega = (0:NF-1) * pi / NF;                     % over the interval [0, π]

Hrec = dtft(hrec, omega);                       % frequency response
Hham = dtft(hham, omega);
```

Problem 10.19

As an example, the following MATLAB program segment computes the bandpass filter of Example 10.2.2 (alternative design):

```
fs = 20;                                        in kHz
fpa = 4; fpb = 6; fsa = 3; fsb = 8;
Apass = 0.1; Astop = 80;                        in dB

h = kbp(fs, fpa, fpb, fsa, fsb, Apass, Astop, -1);   % s = −1 for alternative design

NF = 200;
omega = (0:NF-1) * pi / NF;

H = dtft(h, omega);
```

Similarly, the highpass filter can be designed by:

```
fs = 20;                                        in kHz
fpass = 5; fstop = 4;
Apass = 0.1; Astop = 80;                        in dB

h = klh(-1, fs, fpass, fstop, Apass, Astop);    % s = −1 for highpass

NF = 200;
omega = (0:NF-1) * pi / NF;

H = dtft(h, omega);
```


Problem 10.20

Use the design MATLAB routine `kdiff`, with $f_s = 2$ so that $\omega = 2\pi f/2 = \pi f$, that is, the f's are the ω's in units of π. The filter lengths depend only on the values of Δf and A, they are:

Δf	A	N
0.10	30	33
0.10	60	75
0.05	30	63
0.05	60	147

For example, in the case $\omega_c = 0.4\pi$, $\Delta\omega = 0.05\pi$, $A = 60$ dB, we have the MATLAB code:

```
h = kdiff(2, 0.4, 0.05, 60);

NF = 200;
omega = (0:NF-1) * pi / NF;

H = dtft(h, omega);
```

The magnitude responses are shown in Figs. P10.2 and P10.3.

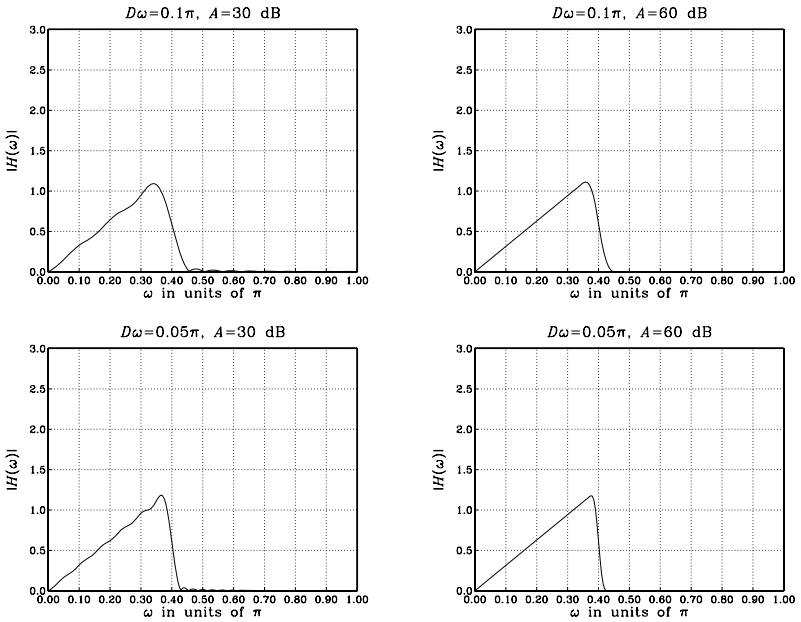


Fig. P10.2 Lowpass differentiator with $\omega_c = 0.4\pi$.

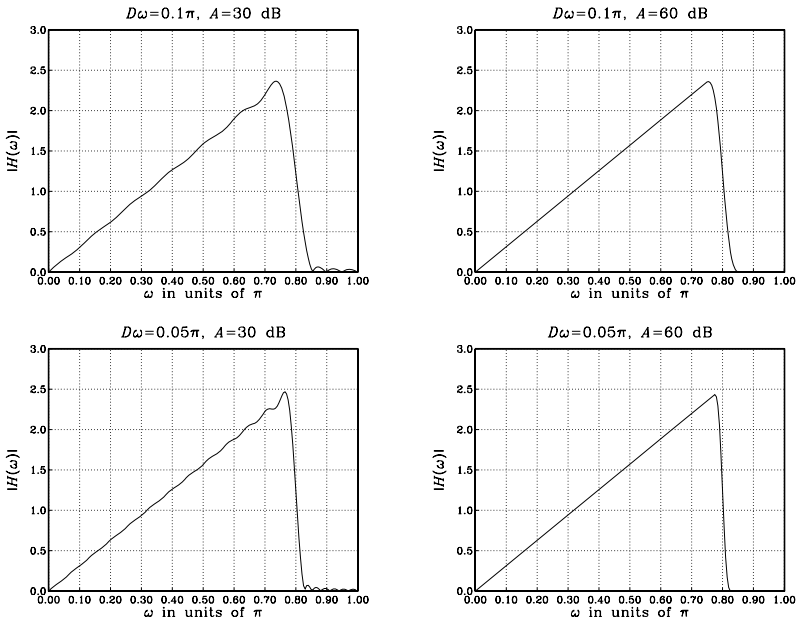


Fig. P10.3 Lowpass differentiator with $\omega_c = 0.8\pi$.

Problem 10.21

For example, in the case $\omega_c = 0.4\pi$, $\Delta\omega = 0.05\pi$, $A = 60$ dB, we have the MATLAB segment that computes the Kaiser differentiator and the Savitzky-Golay differentiators of polynomial order $d = 3$:

```
h = kdiff(2, 0.4, 0.05, 60);                % Kaiser differentiator

[B, S] = sg(3, N);                          % Savitzky-Golay smoothing filters
F = S' * S;                                 % polynomial basis matrix
G = S * F^(-1);                             % Savitzky-Golay differentiator filters
g = G(:,2)';                                % first-order derivative filter

NF = 500;
omega = (0:NF-1) * pi / NF;

H = dtft(h, omega);                         % frequency responses
G = dtft(g, omega);
```

The magnitude responses are shown in Fig. P10.4.

Problem 10.22

Use the design MATLAB routine `khilb`, with $f_s = 2$ so that $\omega = 2\pi f/2 = \pi f$, that is, the f's are the ω's in units of π. The filter lengths depend only on the values of Δf and A, they are the

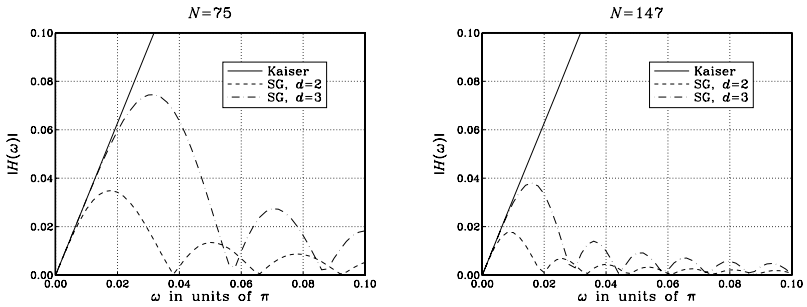


Fig. P10.4 Comparison of Kaiser and SG differentiators.

same as in Problem 10.20. For example, the Hilbert transformer with $\omega_c = 0.8\pi$, $\Delta\omega = 0.05\pi$, and $A = 60$ dB, can be designed by the MATLAB segment:

```
h = khilb(2, 0.8, 0.05, 60);

NF = 200;
omega = (0:NF-1) * pi / NF;

H = dtft(h, omega);                         % frequency responses
```

The magnitude responses are shown in Figs. P10.5 and P10.6.

Problem 10.23

The following MATLAB segment computes the Kaiser and Hamming spectra in dB. The Kaiser window parameters are determined by the function `kparm2.m`:

```
F1 = 0.20; F2 = 0.25; F3 = 0.30; DF = (F2-F1)/3;    % units of fs
A1 = 1; A2 = 10^(-50/20); A3 = 1;                   % amplitudes
R = 70;                                             % sidelobe level in dB

[alpha, L] = kparm2(DF, R);                         % Kaiser parameters

n = (0:L-1);

x = A1 * cos(2*pi*F1*n) + A2 * cos(2*pi*F2*n) + A3 * cos(2*pi*F3*n);

xk = kwind(alpha, L) .* x;                          % Kaiser windowed signal

xh = (0.54 - 0.46 * cos(2*pi*n/(L-1))) .* x;        % Hamming windowed signal

NF = 256;
omega = (0:NF-1) * pi / NF;

Xk = abs(dtft(xk, omega));                          % DTFT spectrum
Xkmax = max(Xk);
Xk = 20 * log10(Xk / Xkmax);                        % normalized and in dB

Xh = abs(dtft(xh, omega));
```

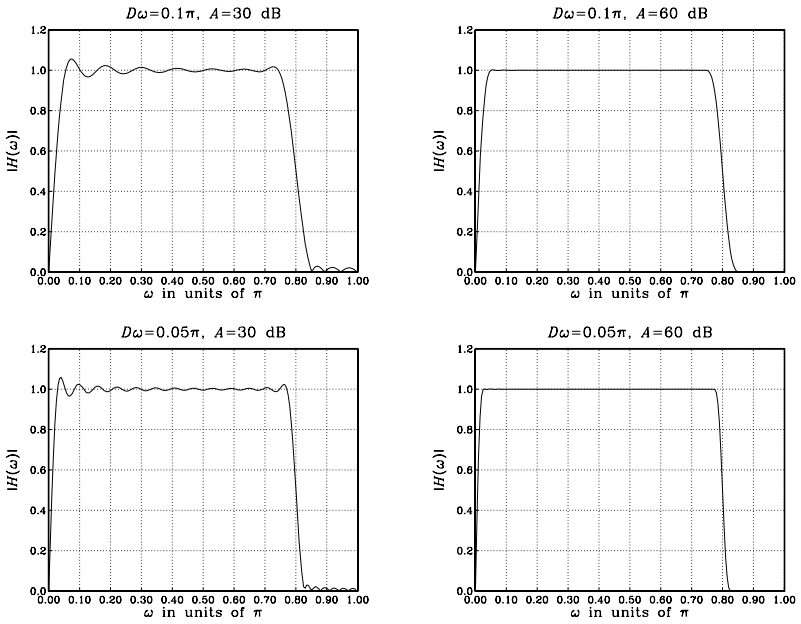


Fig. P10.5 Lowpass Hilbert transformer with $\omega_c = 0.8\pi$.

```
Xhmax = max(Xh);
Xh = 20 * log10(Xh / Xhmax);
```

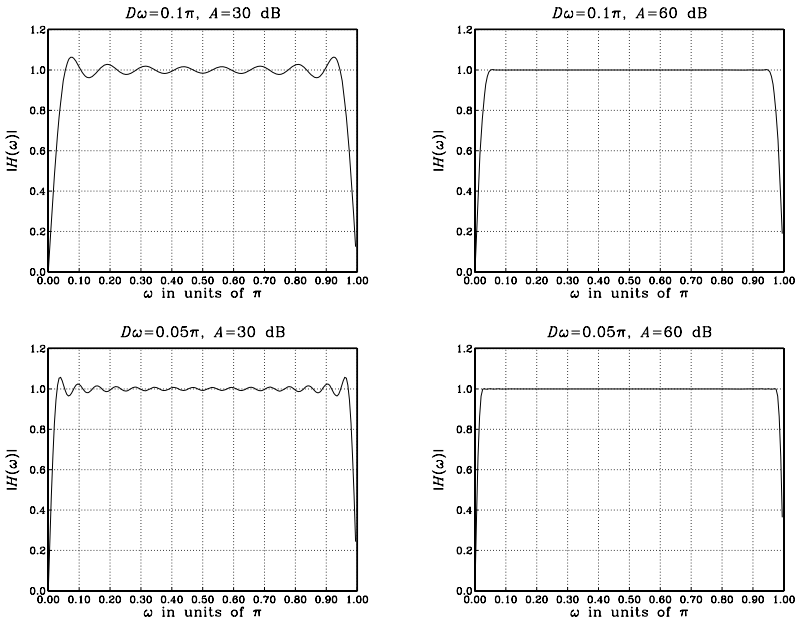



Fig. P10.6 Full-band Hilbert transformer.

Chapter 11 Problems

Problem 11.1

Here, we assume that the parameter $\beta = \tan(\Delta\omega/2)$ remains positive and less than one, $0 \le \beta \le 1$. This is guaranteed if the bandwidth is $0 \le \Delta\omega \le \pi/2$ in rads/sample, or, $0 \le \Delta f \le f_s/4$ in Hz. This range covers most practical applications. This condition implies that the poles are complex conjugate and have radius

$$R^2 = \frac{1-\beta}{1+\beta} \qquad \Rightarrow \qquad R = \sqrt{\frac{1-\beta}{1+\beta}}$$

(If $\beta > 1$, which happens when $\pi/2 < \Delta\omega \le \pi$, then there are two real poles.) It follows that the ϵ-level time constant will be

$$n_{\text{eff}} = \frac{\ln\epsilon}{\ln R}$$

Problem 11.2

For small values of $\Delta\omega$, we may use the first-order Taylor expansion $\tan(x) \simeq x$ to write approximately:

$$\beta = \tan\left(\frac{\Delta\omega}{2}\right) \simeq \frac{\Delta\omega}{2}$$

Then, expanding R to first-order in β and thus to first-order in $\Delta\omega$, we have:

$$R = \sqrt{\frac{1-\beta}{1+\beta}} \simeq (1-\frac{1}{2}\beta)\,(1-\frac{1}{2}\beta) \simeq 1-\beta \simeq 1-\frac{\Delta\omega}{2}$$

where we used the Taylor expansions $(1 \mp x)^{\pm 1/2} \simeq 1 - x/2$ and $(1-x/2)^2 \simeq 1-x$. Solving for $\Delta\omega$, we obtain the approximate expression $\Delta\omega = 2(1-R)$.

Problem 11.3

For the 3-dB case, the parameter b is the same for both filters. Adding the two transfer functions, we recognize that the two numerators add up to become the denominator:

$$\begin{aligned} H_{\text{notch}}(z) + H_{\text{peak}}(z) &= b\,\frac{1-2\cos\omega_0\,z^{-1}+z^{-2}}{1-2b\cos\omega_0\,z^{-1}+(2b-1)z^{-2}} + (1-b)\,\frac{1-z^{-2}}{1-2b\cos\omega_0\,z^{-1}+(2b-1)z^{-2}} \\ &= \frac{b(1-2\cos\omega_0\,z^{-1}+z^{-2}) + (1-b)(1-z^{-2})}{1-2b\cos\omega_0\,z^{-1}+(2b-1)z^{-2}} = \frac{1-2b\cos\omega_0\,z^{-1}+(2b-1)z^{-2}}{1-2b\cos\omega_0\,z^{-1}+(2b-1)z^{-2}} = 1 \end{aligned}$$

For the magnitude responses squared we may use the equivalent analog expressions to get:

$$|H_{\text{notch}}(\omega)|^2 + |H_{\text{peak}}(\omega)|^2 = |H_{\text{notch}}(\Omega)|^2 + |H_{\text{peak}}(\Omega)|^2 = \frac{(\Omega^2-\Omega_0^2)^2}{(\Omega^2-\Omega_0^2)^2+\alpha^2\Omega^2} + \frac{\alpha^2\Omega^2}{(\Omega^2-\Omega_0^2)^2+\alpha^2\Omega^2} = 1$$

Problem 11.4

The analog transfer function and magnitude response squared are in the general boost or cut case:

$$H(s) = \frac{G_0(s^2+\Omega_0^2)+G\alpha s}{s^2+\alpha s+\Omega_0^2} \qquad \Rightarrow \qquad |H(\Omega)|^2 = \frac{G_0^2(\Omega^2-\Omega_0^2)^2+G^2\alpha^2\Omega^2}{(\Omega^2-\Omega_0^2)^2+\alpha^2\Omega^2}$$

where

$$\alpha = (1+\Omega_0^2)\beta = (1+\Omega_0^2)\tan\left(\frac{\Delta\omega}{2}\right)\sqrt{\frac{G_B^2-G_0^2}{G^2-G_B^2}} \equiv \gamma\sqrt{\frac{G_B^2-G_0^2}{G^2-G_B^2}}$$

where the last step defines the parameter γ, which does not depend on the gains. Normalizing everything to the reference gain G_0, we have

$$|H(\Omega)|^2 = G_0^2\,\frac{(\Omega^2-\Omega_0^2)^2+g^2\alpha^2\Omega^2}{(\Omega^2-\Omega_0^2)^2+\alpha^2\Omega^2}, \qquad \alpha = \gamma\sqrt{\frac{g_B^2-1}{g^2-g_B^2}}, \qquad \text{where} \qquad g = \frac{G}{G_0}, \quad g_B = \frac{G_B}{G_0}$$

Then, we may express the magnitude squared response in the following form that explicitly shows the gains:

$$|H(\Omega)|^2 = G_0^2\,\frac{(\Omega^2-\Omega_0^2)^2+N(g,g_B)\gamma^2\Omega^2}{(\Omega^2-\Omega_0^2)^2+D(g,g_B)\gamma^2\Omega^2}$$

where the numerator and denominator gain functions are defined by

$$N(g,g_B) = g^2\,\frac{g_B^2-1}{g^2-g_B^2}, \qquad D(g,g_B) = \frac{g_B^2-1}{g^2-g_B^2}$$

For the arithmetic, geometric, and harmonic mean choices for g_B^2 the functions N and D simplify as follows:

$$\begin{aligned} &g_B^2 = \frac{1+g^2}{2}, \quad N(g,g_B) = g^2, \quad D(g,g_B) = 1, && \text{(arithmetic)} \\ &g_B^2 = g, \quad N(g,g_B) = g, \quad D(g,g_B) = \frac{1}{g}, && \text{(geometric)} \\ &g_B^2 = \frac{2g^2}{1+g^2}, \quad N(g,g_B) = 1, \quad D(g,g_B) = \frac{1}{g^2}, && \text{(harmonic)} \end{aligned}$$

Now, if we consider an equal and opposite boost and cut by, say by A dB (with respect to the reference), then the boost and cut gains will be $g = 10^{\pm A/20}$. Thus, the substitution $g \to g^{-1}$ changes a boost into an equal and opposite cut, and vice versa. Even though the boost and cut gains are equal and opposite, the bandwidth levels g_B^2 may be defined arbitrarily in the two cases. However, if we take them also to be related by the substitution $g_B \to g_B^{-1}$, then we can show that the boost and cut filters are inverses to each other. Indeed, under the substitutions $\{g, g_B\} \to \{g^{-1}, g_B^{-1}\}$, the boost and cut magnitude responses are:

$$|H_{\text{boost}}(\Omega)|^2 = G_0^2\,\frac{(\Omega^2-\Omega_0^2)^2+N(g,g_B)\gamma^2\Omega^2}{(\Omega^2-\Omega_0^2)^2+D(g,g_B)\gamma^2\Omega^2}, \qquad |H_{\text{cut}}(\Omega)|^2 = G_0^2\,\frac{(\Omega^2-\Omega_0^2)^2+N(g^{-1},g_B^{-1})\gamma^2\Omega^2}{(\Omega^2-\Omega_0^2)^2+D(g^{-1},g_B^{-1})\gamma^2\Omega^2}$$

Under this substitution, the functions N and D interchange roles, that is, one can show easily that

$$N(g^{-1},g_B^{-1}) = D(g,g_B), \qquad D(g^{-1},g_B^{-1}) = N(g,g_B)$$

Therefore, the boost and cut filters will be:

$$|H_{\text{boost}}(\Omega)|^2 = G_0^2\,\frac{(\Omega^2-\Omega_0^2)^2+N(g,g_B)\gamma^2\Omega^2}{(\Omega^2-\Omega_0^2)^2+D(g,g_B)\gamma^2\Omega^2}\,, \qquad |H_{\text{cut}}(\Omega)|^2 = G_0^2\,\frac{(\Omega^2-\Omega_0^2)^2+D(g,g_B)\gamma^2\Omega^2}{(\Omega^2-\Omega_0^2)^2+N(g,g_B)\gamma^2\Omega^2}$$

Hence, their product will be identically equal to G_0^4. The geometric-mean and weighted geometric-mean choices for g_B^2 are special cases of the substitution $\{g, g_B\} \to \{g^{-1}, g_B^{-1}\}$, and therefore, satisfy the desired property. But, any other choice of g_B will do, as long as one defines the boost bandwidth to be g_B and the cut bandwidth, g_B^{-1}, that is, equal and opposite bandwidth gains in dB. Note, also that the transfer functions themselves are inverses to each other, not just their magnitudes. Indeed, we can write them in the form:

$$H_{\text{boost}}(s) = G_0\,\frac{s^2+\Omega_0^2+\gamma s\sqrt{N(g,g_B)}}{s^2+\Omega_0^2+\gamma s\sqrt{D(g,g_B)}}\,, \qquad H_{\text{cut}}(s) = G_0\,\frac{s^2+\Omega_0^2+\gamma s\sqrt{N(g^{-1},g_B^{-1})}}{s^2+\Omega_0^2+\gamma s\sqrt{D(g^{-1},g_B^{-1})}} = G_0\,\frac{s^2+\Omega_0^2+\gamma s\sqrt{D(g,g_B)}}{s^2+\Omega_0^2+\gamma s\sqrt{N(g,g_B)}}$$

which implies for the analog and the bilinearly transformed digital transfer functions:

$$H_{\text{cut}}(s) = \frac{G_0^2}{H_{\text{boost}}(s)} \qquad \Rightarrow \qquad H_{\text{cut}}(z) = \frac{G_0^2}{H_{\text{boost}}(z)}$$

Problem 11.5

The following MATLAB code segment designs the three filters (two peaking and one complementary notch), computes their impulse responses, and their magnitude responses:

```
w0 = 0.35*pi; Dw = 0.1*pi;                                  in rads/sample

[b1, a1, beta1] = parmeq(0, 1, 1/sqrt(2), w0, Dw);          peaking 3-dB width
[b2, a2, beta2] = parmeq(0, 1, sqrt(0.1), w0, Dw);          peaking 10-dB width
[b3, a3, beta3] = parmeq(1, 0, 1/sqrt(2), w0, Dw);          notching 3-dB width

x = [1, zeros(1, 99)];                                      unit impulse

h1 = filter(b1, a1, x);                                     impulse responses
h2 = filter(b2, a2, x);
h3 = filter(b3, a3, x);

w = (0:199)*pi/200;                                         200 frequencies over [0,π]
H1 = abs(dtft(b1, w) ./ dtft(a1, w)).^2;                    magnitude responses
H2 = abs(dtft(b2, w) ./ dtft(a2, w)).^2;
H3 = abs(dtft(b3, w) ./ dtft(a3, w)).^2;
```

The 5-percent time constants can be calculated as in Problem 11.1, with $\epsilon = 0.05$.

Problem 11.6

For example, the following MATLAB code designs the first four filters of Example 11.4.1 and computes their impulse response, frequency, and phase responses (in degrees):

```
[b1, a1, beta1] = parmeq(1, 10^(9/20),  10^(6/20),  0.35*pi, 0.1*pi);
[b2, a2, beta2] = parmeq(1, 10^(9/20),  10^(3/20),  0.35*pi, 0.1*pi);
[b3, a3, beta3] = parmeq(1, 10^(-9/20), 10^(-6/20), 0.60*pi, 0.2*pi);
[b4, a4, beta4] = parmeq(1, 10^(-9/20), 10^(-3/20), 0.60*pi, 0.2*pi);

x = [1, zeros(1, 49)];                                      length-50 unit impulse

h1 = filter(b1, a1, x);                                     impulse responses
h2 = filter(b2, a2, x);
h3 = filter(b3, a3, x);
h4 = filter(b4, a4, x);

w = (0:199)*pi/200;
H1 = dtft(b1, w) ./ dtft(a1, w);  argH1 = angle(H1) * 180 / pi;
H2 = dtft(b2, w) ./ dtft(a2, w);  argH2 = angle(H2) * 180 / pi;
H3 = dtft(b3, w) ./ dtft(a3, w);  argH3 = angle(H3) * 180 / pi;
H4 = dtft(b4, w) ./ dtft(a4, w);  argH4 = angle(H4) * 180 / pi;
```

Figure P11.1 shows the phase responses. Note the rapid variations in the vicinities of the center frequencies.

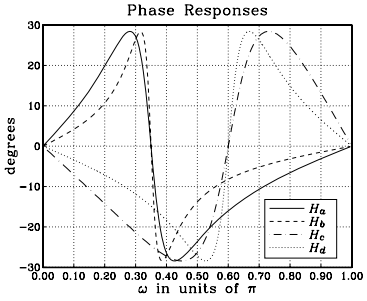


Fig. P11.1 Phase responses of Problem 11.6.

Problem 11.7

The following MATLAB function `lheq.m` implements the design of both the lowpass and highpass shelving filters (the parameter $s = \pm 1$ selects the two cases):

```
% lheq.m - lowpass/highpass first-order shelving EQ filter design
%
% [b, a, beta] = lheq(s, G0, G, Gc, wc)
%
% s  = 1, −1  = lowpass, highpass
% b  = [b0, b1] = numerator coefficients
% a  = [1,  a1] = denominator coefficients
% G0, G, Gc  = reference, boost/cut, and cutoff gains
% wc  = cutoff frequency in [rads/sample]
% beta = design parameter β

function [b, a, beta] = lheq(s, G0, G, Gc, wc)

beta = (tan(wc/2))^s * sqrt(abs(Gc^2 - G0^2)) / sqrt(abs(G^2 - Gc^2));
b = [(G0 + G*beta), -s*(G0 - G*beta)] / (1+beta);
a = [1, -s*(1-beta)/(1+beta)];
```

The filters of Examples 11.2.1 and 11.2.2 may be designed by the following MATLAB code, which also shows how to compute the magnitude responses of the two complementary cases:

```
[b1, a1, beta1] = lheq(1, 0, 1, 1/sqrt(2), 0.2*pi);      lowpass cases
[b2, a2, beta2] = lheq(1, 0, 1, sqrt(0.9), 0.2*pi);

[b3, a3, beta3] = lheq(1, 0, 1, 1/sqrt(2), 0.7*pi);
[b4, a4, beta4] = lheq(1, 0, 1, sqrt(0.9), 0.7*pi);

[b5, a5, beta5] = lheq(-1, 0, 1, 1/sqrt(2), 0.2*pi);     highpass cases
[b6, a6, beta6] = lheq(-1, 0, 1, sqrt(0.9), 0.2*pi);

omega = (0:199)*pi/200;                                  200 frequencies in [0,π]

H1 = abs(dtft(b1, omega)./dtft(a1, omega)).^2;           magnitude response squared
H5 = abs(dtft(b5, omega)./dtft(a5, omega)).^2;
```

Problem 11.8

The following MATLAB code generates the first pair of comb and notch filters of Example 11.5.1, that is, with unshifted peaks, and computes the corresponding magnitude and phase responses; it also computes the impulse response of the first filter in two ways: (a) using MATLAB's function `filter` and (b) the customized function `combfilt.m` that uses a circular buffer:

```
D = 10;
wc = 0.025 * pi;

[a1, b1, c1, beta1] = combeq(1, 10^(9/20),   10^(3/20),  D, wc, 1);
[a2, b2, c2, beta2] = combeq(1, 10^(-12/20), 10^(-3/20), D, wc, 1);

omega = (0:399)*2*pi/ 400;                               400 frequencies in [0,2π]

num1  = [b1, zeros(1, D-1), -c1];                        numerator coefficients
den1  = [1,  zeros(1, D-1), -a1];                        denominator coefficients
H1    = dtft(num1, omega) ./ dtft(den1, omega);          frequency response
magH1 = 20 * log10(abs(H1));                             magnitude response
angH1 = angle(H1) * 180 / pi;                            phase response in degrees

num2  = [b2, zeros(1, D-1), -c2];
den2  = [1,  zeros(1, D-1), -a2];
H2    = dtft(num2, omega) ./ dtft(den2, omega);
magH2 = 20 * log10(abs(H2));
angH2 = angle(H2) * 180 / pi;

x  = [1, zeros(1, 199)];                                 length-200 unit impulse
h1 = filter(num1, den1, x);                              impulse response

w = zeros(1, D+1);                                       (D+1)-dimensional state vector
q = 0;                                                   circular pointer index

for n=0:199,
       [y1(n+1), w, q] = combfilt(a1, b1, c1, D, w, q, x(n+1));      note: h1=y1
end
```

The phase responses of the comb and notch filters and the impulse response of the comb are shown in Fig. P11.2.

The canonical realization and the corresponding circular buffer version of the sample processing algorithm of the transfer function $H(z) = \dfrac{b - cz^{-D}}{1 - az^{-D}}$ are given below:

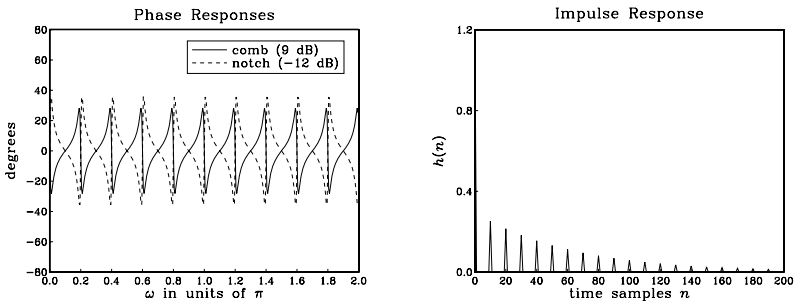
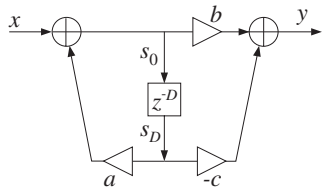


Fig. P11.2 Phase responses and impulse response of Problem 11.8.

for each input sample x do:
$s_D = \text{tap}(D, \mathbf{w}, p, D)$
$s_0 = x + as_D$
$y = bs_0 - cs_D$
$*p = s_0$
$\text{cdelay}(D, \mathbf{w}, \&p)$



The following C function `combfilt.c` is an implementation:

```
/* combfilt.c - comb filtering with circular buffer */

void cdelay();
double tap();

double combfilt(a, b, c, D, w, p, x)
double a, b, c, *w, **p, x;              p passed by address as in cfir.c
int D;
{
       double s0, sD, y;

       sD = tap(D, w, *p, D);            Dth tap output
       s0 = x + a * sD;                  input to D-fold delay
       y = b * s0 - c * sD;              output sample
       **p = s0;                         update circular delay line
       cdelay(D, w, p);

       return y;
}
```

The MATLAB version is the function `combfilt.m` given below:

```
% combfilt.m - comb filtering with circular buffer
%
% [y, w, q] = combfilt(a, b, c, D, w, q, x);
%
% The filter parameters [a, b, c] can be obtained from combeq.m
```

```
%
% It uses cdelay2.m, wrap2.m, tap2.m
% based on combfilt.c

function [y, w, q] = combfilt(a, b, c, D, w, q, x)

sD = tap2(D, w, q, D);
s0 = x + a * sD;
y = b * s0 - c * sD;
w(q+1) = s0;
q = cdelay2(D, q);
```

It uses the following routine `tap2.m` that helps extract the components of the circular internal state vector:

```
% tap2.m - read ith tap of circular delay-line buffer
%
% si = tap2(D, w, q, i)
%
% w = (D+1)-dimensional
% q = circular index into w
% delay amount i = 0, 1, ..., D
% based on tap2.c

function si = tap2(D, w, q, i)

si = w(rem(q+i, D+1)+1);                    % s_i is delayed output s(n-i)
```

Problem 11.9

In the FIR case of designing a lowpass filter, the limits $1 \pm \delta_{\text{pass}}$ and $\pm\delta_{\text{stop}}$ represent the ripples of the (real-valued) frequency response $\hat{D}(\omega)$. These translate to the limits for the magnitude response $1 \pm \delta_{\text{pass}}$ in the passband and $[0, \delta_{\text{stop}}]$ in the stopband.

In the IIR case, the definitions are motivated by the closed-form expressions of the magnitude response squared in the Butterworth and Chebyshev cases.

Problem 11.10

Because $\Omega_{\text{stop}} > \Omega_{\text{pass}}$, and $N \geq N_{\text{exact}}$, we have the inequality

$$\left(\frac{\Omega_{\text{stop}}}{\Omega_{\text{pass}}}\right)^{2N} > \left(\frac{\Omega_{\text{stop}}}{\Omega_{\text{pass}}}\right)^{2N_{\text{exact}}} = \frac{\varepsilon_{\text{stop}}^2}{\varepsilon_{\text{pass}}^2} \qquad \Rightarrow \qquad 1 + \varepsilon_{\text{pass}}^2 \left(\frac{\Omega_{\text{stop}}}{\Omega_{\text{pass}}}\right)^{2N} > 1 + \varepsilon_{\text{stop}}^2$$

Thus, using Eq. (11.6.14) we have:

$$A(\Omega_{\text{stop}}) = 10\log_{10}\left[1 + \varepsilon_{\text{pass}}^2 \left(\frac{\Omega_{\text{stop}}}{\Omega_{\text{pass}}}\right)^{2N}\right] > 10\log_{10}(1 + \varepsilon_{\text{stop}}^2) = A_{\text{stop}}$$

Similarly, if we fix Ω_0 by the stopband specification, then we have:

$$\left(\frac{\Omega_{\text{pass}}}{\Omega_{\text{stop}}}\right)^{2N} < \left(\frac{\Omega_{\text{pass}}}{\Omega_{\text{stop}}}\right)^{2N_{\text{exact}}} = \frac{\varepsilon_{\text{pass}}^2}{\varepsilon_{\text{stop}}^2} \qquad \Rightarrow \qquad 1 + \varepsilon_{\text{stop}}^2 \left(\frac{\Omega_{\text{pass}}}{\Omega_{\text{stop}}}\right)^{2N} < 1 + \varepsilon_{\text{pass}}^2$$

Then, using Eq. (11.6.15), we have:

$$A(\Omega_{\text{pass}}) = 10\log_{10}\left[1 + \varepsilon_{\text{stop}}^2 \left(\frac{\Omega_{\text{pass}}}{\Omega_{\text{stop}}}\right)^{2N}\right] < 10\log_{10}(1 + \varepsilon_{\text{pass}}^2) = A_{\text{pass}}$$

Thus, in either case, the specifications are more than satisfied.

Problem 11.11

The call to the MATLAB function `lhbutt.m`:

```
[A, B, P] = lhbutt(1, 40, 10, 15, 3, 35);
```

produces the following cascade matrices

$$B = \begin{bmatrix} 0.5001 & 0.5001 & 0 \\ 0.3821 & 2\times 0.3821 & 0.3821 \\ 0.2765 & 2\times 0.2765 & 0.2765 \end{bmatrix}, \qquad A = \begin{bmatrix} 1 & 0.0002 & 0 \\ 1 & 0.0007 & 0.5279 \\ 1 & 0.0005 & 0.1056 \end{bmatrix}$$

Thus, the designed transfer function will be:

$$H(z) = \frac{0.5001(1+z^{-1})}{1+0.0002z^{-1}} \cdot \frac{0.3821(1+z^{-1})^2}{1+0.0007z^{-1}+0.5279z^{-2}} \cdot \frac{0.2765(1+z^{-1})^2}{1+0.0005z^{-1}+0.1056z^{-2}}$$

The parameter vector P is:

$$\begin{aligned} P &= [\Omega_{\text{pass}}, \Omega_{\text{stop}}, \varepsilon_{\text{pass}}, \varepsilon_{\text{stop}}, N_{\text{exact}}, N, A_{\text{stop}}, \Omega_0, f_0] \\ &= [1,\ 2.4142,\ 0.9976,\ 56.2252,\ 4.57,\ 5,\ 38.26,\ 1.0005,\ 10.0030] \end{aligned}$$

where A_{stop} is the actual stopband attenuation realized by the designed filter, and f_0 is the 3-dB cutoff frequency. The apparent discrepancy between the computed 3-dB frequency and the given 3-dB passband specification is accounted for from the fact that the 3-dB frequency Ω_0 or f_0 is defined to be the frequency where the magnitude squared response drops to one-half its DC value; this is almost a 3-dB drop, but not exactly. By hand, the design is as follows. First, we prewarp the physical frequencies:

$$\Omega_{\text{pass}} = \tan\left(\frac{\omega_{\text{pass}}}{2}\right) = \tan\left(\frac{\pi f_{\text{pass}}}{f_s}\right) = \tan\left(\frac{\pi 10}{40}\right) = \tan(0.25\pi) = 1$$

$$\Omega_{\text{stop}} = \tan\left(\frac{\omega_{\text{stop}}}{2}\right) = \tan\left(\frac{\pi f_{\text{stop}}}{f_s}\right) = \tan\left(\frac{\pi 15}{40}\right) = \tan(0.375\pi) = 2.4142$$

Then, we compute the ϵ-parameters:

$$\varepsilon_{\text{pass}} = \sqrt{10^{A_{\text{pass}}/10} - 1} = \sqrt{10^{3/10} - 1} = 0.9976, \qquad \varepsilon_{\text{stop}} = \sqrt{10^{A_{\text{stop}}/10} - 1} = \sqrt{10^{35/10} - 1} = 56.2252$$

and the exact and up-rounded value of N:

$$N_{\text{exact}} = \frac{\ln(\varepsilon_{\text{stop}}/\varepsilon_{\text{pass}})}{\ln(\Omega_{\text{stop}}/\Omega_{\text{pass}})} = \frac{\ln(56.2252/0.9976)}{\ln(2.4142/1.0000)} = 4.57\,, \qquad N = 5$$

Then, we calculate the 3-dB Butterworth parameter Ω_0 and invert it to get the 3-dB frequency f_0:

$$\Omega_0 = \frac{\Omega_{\text{pass}}}{\varepsilon_{\text{pass}}^{1/N}} = \frac{1}{(0.9976)^{1/5}} = 1.0005\,, \qquad f_0 = \frac{f_s}{\pi}\arctan(\Omega_0) = 1.0030 \text{ kHz}$$

The coefficients of the 2nd-order sections are computed by Eq. (11.6.24). Note that because of the small discrepancy mentioned above, Ω_0 is not exactly equal to unity, and therefore, the a_1 coefficients of all the sections will not be exactly zero. The Butterworth angles are computed by

$$\theta_i = \frac{\pi}{2N}(N-1+2i) = \frac{\pi}{10}(4+2i), \qquad i = 1,2$$

which gives

$$\theta_1 = 0.6\pi, \qquad \theta_2 = 0.8\pi$$

Problem 11.12

The call to `lhbutt.m`:

```
[A, B, P] = lhbutt(-1, 10, 3, 2, 0.5, 35);
```

produces the following cascade matrices

$$B = \begin{bmatrix} 0.3414 & -2\times 0.3414 & 0.3414 \\ 0.2454 & -2\times 0.2454 & 0.2454 \end{bmatrix}, \qquad A = \begin{bmatrix} 1 & 0.0817 & 0.4471 \\ 1 & 0.0587 & 0.0404 \end{bmatrix}$$

Thus, the designed transfer function will be:

$$H(z) = \frac{0.3414(1-z^{-1})^2}{1+0.0817z^{-1}+0.4471z^{-2}} \cdot \frac{0.2454(1-z^{-1})^2}{1+0.0587z^{-1}+0.0404z^{-2}}$$

The parameter vector P is:

$$P = [\Omega_{\text{pass}}, \Omega_{\text{stop}}, \varepsilon_{\text{pass}}, \varepsilon_{\text{stop}}, N_{\text{exact}}, N, A_{\text{stop}}, \Omega_0, f_0]$$

$$= [0.7265,\ 1.3764,\ 0.3493,\ 3.0000,\ 3.37,\ 4,\ 13.27,\ 0.9451,\ 2.5899]$$

where A_{stop} is the actual stopband attenuation realized by the designed filter, and f_0 is the 3-dB cutoff frequency in kHz. The required closed-form expression for part (c) is:

$$|H(\omega)|^2 = \frac{1}{1+\left(\dfrac{\cot(\omega/2)}{\Omega_0}\right)^{2N}}$$

Problem 11.13

The real part of s can be expressed as follows:

$$\text{Re}\, s = \frac{1}{2}(s+s^*) = \frac{z^2-2cz+1}{2(z^2-1)} + \frac{z^{*2}-2cz^*+1}{2(z^{*2}-1)} = \frac{(z^2-2cz+1)(z^{*2}-1)+(z^{*2}-2cz^*+1)(z^2-1)}{2(z^2-1)(z^2-1)^*}$$

$$= \frac{|z|^4-1-c(z+z^*)(|z|^2-1)}{|z^2-1|^2} = \frac{(|z|^2-1)(|z|^2-c(z+z^*)+1)}{|z^2-1|^2}$$

where in the last step we factored $|z|^4-1 = (|z|^2-1)(|z|^2+1)$. Because $|c| \le 1$, the second factor of the numerator is non-negative. Indeed, noting that $(z+z^*)$ is real-valued, we have:

$$|c(z+z^*)| \le |z+z^*| \le |z|+|z^*| = 2|z| \qquad \Rightarrow \qquad -2|z| \le c(z+z^*) \le 2|z|$$

Adding $|z|^2+1$ to both sides we get the inequality:

$$|z|^2-2|z|+1 \le |z|^2-c(z+z^*)+1 \le |z|^2+2|z|+1 \qquad \Rightarrow \qquad 0 \le (|z|-1)^2 \le |z|^2-c(z+z^*)+1 \le (|z|+1)^2$$

Therefore, the sign of $\text{Re}\, s$ is the same as the sign of the factor $(|z|^2-1)$, and the desired result follows.

Problem 11.14

Using a trig identity, we obtain:

$$|\sin(\omega_1+\omega_2)| = |\sin\omega_1\cos\omega_2 + \sin\omega_2\cos\omega_1| \le |\sin\omega_1||\cos\omega_2| + |\sin\omega_2||\cos\omega_1|$$

$$\le |\sin\omega_1| + |\sin\omega_2| = \sin\omega_1 + \sin\omega_2$$

where we used the inequalities $|\cos\omega_1| \le 1$ and $|\cos\omega_2| \le 1$ and the fact that $\sin\omega_1$ and $\sin\omega_2$ are non-negative for values of their arguments in the range $[0,\pi]$. Thus, the above inequality implies $|c| \le 1$.

Problem 11.15

Here, the Butterworth filter order is given to be $N = 3$. Thus, there remains only the parameter Ω_0 to be fixed. The normalized passband frequencies are

$$\omega_{pa} = \frac{2\pi f_{pa}}{f_s} = \frac{2\pi 2}{20} = 0.2\pi, \qquad \omega_{pb} = \frac{2\pi f_{pb}}{f_s} = \frac{2\pi 8}{20} = 0.8\pi$$

Because they add up to π, the bilinear transformation parameter c will be zero, as follows from the equation

$$c = \frac{\sin(\omega_{pa}+\omega_{pb})}{\sin\omega_{pa}+\sin\omega_{pb}} = 0$$

Thus, the bilinear transformation will be here:

$$s = \frac{1+z^{-2}}{1-z^{-2}}$$

The analog passband frequency will be:

$$\Omega_{\text{pass}} = \frac{c-\cos\omega_{pb}}{\sin\omega_{pb}} = -\cot\omega_{pb} = 1.3764$$

The $\varepsilon_{\text{pass}}$ parameter is calculated to be:

$$\varepsilon_{\text{pass}} = \sqrt{10^{A_{\text{pass}}/10}-1} = \sqrt{10^{0.5/10}-1} = 0.3493$$

which implies for Ω_0:

$$\Omega_0 = \frac{\Omega_{\text{pass}}}{\varepsilon_{\text{pass}}^{1/N}} = \frac{1.3764}{0.3493^{1/3}} = 1.9543$$

The coefficients of the second- and fourth-order digital filter sections are calculated from Eqs. (11.6.43) and (11.6.41), where the there is only one Butterworth angle, namely,

$$\theta_1 = \frac{\pi}{2N}(N-1+2) = \frac{2\pi}{3} \qquad \Rightarrow \qquad \cos\theta_1 = -0.5$$

We find for the second-order section:

$$G_0 = 0.6615, \quad a_{01} = 0, \quad a_{02} = -0.3230$$

and the fourth-order section:

$$G_1 = 0.5639, \quad a_{11} = 0, \quad a_{12} = -0.8325, \quad a_{13} = 0, \quad a_{14} = 0.4230$$

resulting in the transfer function:

$$H(z) = \frac{0.6615(1-z^{-2})}{1-0.3230z^{-2}} \cdot \frac{0.5639(1-z^{-2})^2}{1-0.8325z^{-2}+0.4230z^{-4}}$$

The 3-dB frequencies are calculated from Eq. (11.6.44) with $c = 0$. We find:

$$f_{0a} = 1.5054 \text{ kHz}, \qquad f_{0b} = 8.4946 \text{ kHz}$$

Problem 11.16

We have:

$$\frac{1}{1-2\dfrac{s}{\Omega_0}\cos\theta_i+\dfrac{s^2}{\Omega_0^2}}=\frac{\Omega_0^2}{\Omega_0^2-2s\Omega_0\cos\theta_i+s^2}=\frac{\Omega_0^2}{\Omega_0^2-2\left(\dfrac{1-z^{-1}}{1+z^{-1}}\right)\Omega_0\cos\theta_i+\left(\dfrac{1-z^{-1}}{1+z^{-1}}\right)^2}$$

$$=\frac{\Omega_0^2(1+z^{-1})^2}{\Omega_0^2(1+z^{-1})^2-2(1-z^{-1})(1+z^{-1})\Omega_0\cos\theta_i+(1-z^{-1})^2}$$

$$=\frac{\Omega_0^2(1+z^{-1})^2}{\Omega_0^2(1+2z^{-1}+z^{-2})-2(1-z^{-2})\Omega_0\cos\theta_i+(1-2z^{-1}+z^{-2})}$$

$$=\frac{\Omega_0^2(1+z^{-1})^2}{(1-2\Omega_0\cos\theta_i+\Omega_0^2)+2(\Omega_0^2-1)z^{-1}+(1+2\Omega_0\cos\theta_i+\Omega_0^2)z^{-2}}=\frac{G_i(1+z^{-1})^2}{1+a_{i1}z^{-1}+a_{i2}z^{-2}}$$

where the final result was obtained by dividing numerator and denominator by the factor $(1-2\Omega_0\cos\theta_i+\Omega_0^2)$ in order to make the constant coefficient of the denominator unity.

The highpass case is obtained by replacing z^{-1} by $-z^{-1}$, which effectively changes the sign of the a_{i1} coefficient. The bandpass case can be done in a similar, but more tedious, fashion.

Problem 11.17

The right 3-dB frequency is obtained by solving the equation

$$\frac{c-\cos\omega_{0b}}{\sin\omega_{0b}}=\Omega_0$$

Using the trig identities

$$\cos\omega_{0b}=\frac{1-\tan^2(\omega_{0b}/2)}{1+\tan^2(\omega_{0b}/2)},\qquad \sin\omega_{0b}=\frac{2\tan(\omega_{0b}/2)}{1+\tan^2(\omega_{0b}/2)}$$

The above equation reduces to the quadratic equation in the variable $t=\tan(\omega_{0b}/2)$:

$$\frac{c-\dfrac{1-t^2}{1+t^2}}{\dfrac{2t}{1+t^2}}=\Omega_0\qquad\Rightarrow\qquad (1+c)t^2-2\Omega_0 t-(1-c)=0$$

which has positive solution:

$$t=\frac{\Omega_0+\sqrt{\Omega_0^2+(1-c)(1+c)}}{1+c}$$

implying for the 3-dB frequency

$$\tan\left(\frac{\omega_{0b}}{2}\right)=\tan\left(\frac{\pi f_{0b}}{f_s}\right)=\frac{\sqrt{\Omega_0^2+1-c^2}+\Omega_0}{1+c}\qquad\Rightarrow\qquad f_{0b}=\frac{f_s}{\pi}\arctan\left(\frac{\sqrt{\Omega_0^2+1-c^2}+\Omega_0}{1+c}\right)$$

The left 3-dB frequency is obtained by replacing Ω_0 by $-\Omega_0$ in the above solution.

Problem 11.18

Matching the stopband specifications exactly means that the stopband frequencies ω_{sa}, ω_{sb} map onto $\pm\Omega_{\text{stop}}$ and that at Ω_{stop} the attenuation is exactly A_{stop}. The first requirement implies

$$-\Omega_{\text{stop}}=\frac{c-\cos\omega_{sa}}{\sin\omega_{sa}}$$

$$\Omega_{\text{stop}}=\frac{c-\cos\omega_{sb}}{\sin\omega_{sb}}$$

which can be solve for c and Ω_{stop}:

$$c=\frac{\sin(\omega_{sa}+\omega_{sb})}{\sin\omega_{sa}+\sin\omega_{sb}},\qquad \Omega_{\text{stop}}=\left|\frac{c-\cos\omega_{sb}}{\sin\omega_{sb}}\right|$$

Given the value of c, we compute next the two candidates for Ω_{pass}:

$$\Omega_{pa}=\frac{c-\cos\omega_{pa}}{\sin\omega_{pa}},\qquad \Omega_{pb}=\frac{c-\cos\omega_{pb}}{\sin\omega_{pb}}$$

Ideally, we would like have $\Omega_{pb}=\Omega_{\text{pass}}$ and $\Omega_{pa}=-\Omega_{\text{pass}}$. But this is impossible because c has already been fixed. Because the Butterworth magnitude response is a *monotonically decreasing* function of Ω, it is enough to choose the widest of the two passbands defined above. Thus, we define:

$$\Omega_{\text{pass}}=\max(|\Omega_{pa}|,|\Omega_{pb}|)$$

Once we have computed Ω_{pass} and Ω_{stop}, we design our Butterworth analog filter; that is, compute N and Ω_0 from

$$N_{\text{exact}}=\frac{\ln(\varepsilon_{\text{stop}}/\varepsilon_{\text{pass}})}{\ln(\Omega_{\text{stop}}/\Omega_{\text{pass}})},\qquad N=\lceil N_{\text{exact}}\rceil,\qquad \Omega_0=\frac{\Omega_{\text{stop}}}{\varepsilon_{\text{stop}}^{1/N}}$$

where the equation for Ω_0 matches the stopband specification exactly. The digital filter coefficients are then computed in terms of N and Ω_0 as usual. The following MATLAB function `bpbutt2.m` implements the above procedure and is an alternative to `bpsbutt.m`:

```
% bpbutt2.m - bandpass Butterworth digital filter design
%
% [A, B, P] = bpbutt2(fs, fpa, fpb, fsa, fsb, Apass, Astop)
%
% alternative to bpsbutt.m
% it matches stopbands and uses worst passband.
%
% design parameters:
% P = [Wpass, Wstop, Wpa, Wpb, c, fc, epass, estop, Nex, N, Apass, W0, f0a, f0b];
% A,B are Kx5 matrices of cascade of 4th/2nd order sections

function [A, B, P] = bpbutt2(fs, fpa, fpb, fsa, fsb, Apass, Astop)

c = sin(2*pi*(fsa + fsb) / fs) / (sin(2*pi*fsa / fs) + sin(2*pi*fsb / fs));

fc = 0.5 * (fs/pi) * acos(c);

Wstop = abs((c - cos(2*pi*fsb / fs)) / sin(2*pi*fsb / fs));

Wpa = (c - cos(2*pi*fpa / fs)) / sin(2*pi*fpa / fs);
Wpb = (c - cos(2*pi*fpb / fs)) / sin(2*pi*fpb / fs);
```

```
Wpass = max(abs(Wpa), abs(Wpb));

epass = sqrt(10.0^(Apass/10) - 1);
estop = sqrt(10.0^(Astop/10) - 1);

Nex = log(estop/epass) / log(Wstop/Wpass);
N = ceil(Nex);
r = rem(N,2);
K = (N - r) / 2;

W0 = Wstop * (estop^(-1/N));

Apass = 10 * log10(1 + (Wpass/W0)^(2*N));

f0a = (fs/pi) * atan((sqrt(W0^2 - c^2 + 1) - W0)/(c+1));
f0b = (fs/pi) * atan((sqrt(W0^2 - c^2 + 1) + W0)/(c+1));

P = [Wpass, Wstop, Wpa, Wpb, c, fc, epass, estop, Nex, N, Apass, W0, f0a, f0b];

if r==1,
    G = W0 / (1 + W0);
    a1 = -2 * c / (1 + W0);
    a2 = (1 - W0) / (1 + W0);
    B(1,:) = G * [1, 0, -1, 0, 0];
    A(1,:) = [1, a1, a2, 0, 0];
else
    B(1,:) = [1, 0, 0, 0, 0];
    A(1,:) = [1, 0, 0, 0, 0];
end

for i=1:K,
    th = pi * (N - 1 + 2 * i) / (2 * N);
    D = 1 - 2 * W0 * cos(th) + W0^2;
    G = W0^2 / D;
    a1 = 4 * c * (W0 * cos(th) - 1) / D;
    a2 = 2 * (2*c^2 + 1 - W0^2) / D;
    a3 = - 4 * c * (W0 * cos(th) + 1) / D;
    a4 = (1 + 2 * W0 * cos(th) + W0^2) / D;
    B(i+1,:) = G * [1, 0, -2, 0, 1];
    A(i+1,:) = [1, a1, a2, a3, a4];
end
```

Similarly, the function `bpcheb2a.m` is an alternative to the bandpass Chebyshev type-2 design function `bpcheb2.m`:

```
% bpcheb2a.m - alternative bandpass Chebyshev type 2 digital filter design
%
% [A, B, P] = bpcheb2a(fs, fpa, fpb, fsa, fsb, Apass, Astop)
%
% alternative to bpcheb2.m
% matches stopbands instead of passbands
%
% design parameters:
% P = [Wpass, Wstop, Wpa, Wpb, c, epass, estop, Nex, N, f3a, f3b, a];
% A, B are Kx5 matrices of cascade 4th/2nd order sections

function [A, B, P] = bpcheb2a(fs, fpa, fpb, fsa, fsb, Apass, Astop)
```



```
c = sin(2*pi*(fsa + fsb) / fs) / (sin(2*pi*fsa / fs) + sin(2*pi*fsb / fs));

Wstop = abs((c - cos(2*pi*fsb / fs)) / sin(2*pi*fsb / fs));

Wpa = (c - cos(2*pi*fpa / fs)) / sin(2*pi*fpa / fs);
Wpb = (c - cos(2*pi*fpb / fs)) / sin(2*pi*fpb / fs);

Wpass = max(abs(Wpa), abs(Wpb));

epass = sqrt(10.0^(Apass/10) - 1);
estop = sqrt(10.0^(Astop/10) - 1);

Nex = acosh(estop/epass) / acosh(Wstop/Wpass);
N = ceil(Nex);
r = rem(N,2);
K = (N - r) / 2;

a = asinh(estop) / N;

W3 = Wstop / cosh(acosh(estop)/N);
f3a = (fs/pi) * atan((sqrt(W3^2 - c^2 + 1) - W3)/(c+1));
f3b = (fs/pi) * atan((sqrt(W3^2 - c^2 + 1) + W3)/(c+1));

P = [Wpass, Wstop, Wpa, Wpb, c, epass, estop, Nex, N, f3a, f3b, a];

W0 = sinh(a) / Wstop;

if r==1,
    G = 1 / (1 + W0);
    a1 = -2 * c * W0 / (1 + W0);
    a2 = -(1 - W0) / (1 + W0);
    B(1,:) = G * [1, 0, -1, 0, 0];
    A(1,:) = [1, a1, a2, 0, 0];
else
    B(1,:) = [1, 0, 0, 0, 0];
    A(1,:) = [1, 0, 0, 0, 0];
end

for i=1:K,
    th = pi * (N - 1 + 2 * i) / (2 * N);
    Wi = sin(th) / Wstop;                                   % reciprocal of text
    D = 1 - 2 * W0 * cos(th) + W0^2 + Wi^2;
    G = (1 + Wi^2) / D;
    b1 = - 4 * c * Wi^2 / (1 + Wi^2);
    b2 = 2 * (Wi^2 * (2*c*c+1) - 1) / (1 + Wi^2);
    a1 = 4 * c * (W0 * cos(th) - W0^2 - Wi^2) / D;
    a2 = 2 * ((2*c*c + 1)*(W0^2 + Wi^2) - 1) / D;
    a3 = - 4 * c * (W0 * cos(th) + W0^2 + Wi^2) / D;
    a4 = (1 + 2 * W0 * cos(th) + W0^2 + Wi^2) / D;
    B(i+1,:) = G * [1, b1, b2, b1, 1];
    A(i+1,:) = [1, a1, a2, a3, a4];
end
```

Problem 11.19

We work with the type-2 design since it has more complicated numerator. The transformation of the analog second-order sections is as follows:

$$\frac{1+\Omega_i^{-2}s^2}{1-2\Omega_0^{-1}\cos\theta_i\, s+(\Omega_0^{-2}+\Omega_i^{-2})s^2} = \frac{(1+z^{-1})^2+\Omega_i^{-2}(1-z^{-1})^2}{(1+z^{-1})^2-2\Omega_0^{-1}\cos\theta_i(1-z^{-1})(1+z^{-1})+(\Omega_0^{-2}+\Omega_i^{-2})(1-z^{-1})^2}$$

$$= \frac{(1+\Omega_i^{-2})+2(1-\Omega_i^{-2})z^{-1}+(1+\Omega_i^{-2})z^{-2}}{(1-2\Omega_0^{-2}\cos\theta_i+\Omega_0^{-2}+\Omega_i^{-2})+2(1-\Omega_0^{-2}-\Omega_i^{-2})z^{-1}+(1+2\Omega_0^{-2}\cos\theta_i+\Omega_0^{-2}+\Omega_i^{-2})z^{-2}}$$

$$= \frac{G_i(1+b_{i1}z^{-1}+z^{-2})}{1+a_{i1}z^{-1}+a_{i2}z^{-2}}$$

where the final step follows by dividing the numerator by the factor $(1+\Omega_i^{-2})$ and the denominator by $(1-2\Omega_0^{-2}\cos\theta_i+\Omega_0^{-2}+\Omega_i^{-2})$.

Problem 11.20

The s-domain poles are

$$s_i = \Omega_{\text{pass}}\sinh a\,\cos\theta_i + j\Omega_{\text{pass}}\cosh a\,\sin\theta_i = \Omega_0\cos\theta_i + j\Omega_i\cosh a$$

It follows that

$$|s_i|^2 = \Omega_0^2\cos^2\theta_i + \Omega_i{}^2\cosh^2 a = \Omega_0^2(1-\sin^2\theta_i)+\Omega_i{}^2(1+\sinh^2 a)$$

$$= \Omega_0^2+\Omega_i{}^2+(\Omega_0^2\sin^2\theta_i-\Omega_i{}^2\sinh^2 a) = \Omega_0^2+\Omega_i{}^2$$

where the last two terms canceled because

$$\Omega_0\sin\theta_i = \Omega_{\text{pass}}\sinh a\sin\theta_i = \Omega_i\sinh a$$

Problem 11.21

The following C functions `sos4.c` and `cas4.c` implement the sample-by-sample processing algorithm of the cascade of fourth-order sections:

```
/* sos4.c - IIR filtering by single (canonical) fourth-order section */

double sos4(a, b, w, x)                    a,b,w are 5-dimensional
double *a, *b, *w, x;                      a[0] = 1 always
{
       double y;

       w[0] = x - a[1]*w[1] - a[2]*w[2] - a[3]*w[3] - a[4]*w[4];
       y = b[0]*w[0] + b[1]*w[1] + b[2]*w[2] + b[3]*w[3] + b[4]*w[4];

       w[4] = w[3];                        could replace these updates by
       w[3] = w[2];                        the single call to
       w[2] = w[1];                        delay(4, w);
       w[1] = w[0];

       return y;
}
```

and


```
/* cas4.c - IIR filtering in cascade of fourth-order sections */

double sos4();                                   single 4th-order section

double cas4(K, A, B, W, x)
int K;
double **A, **B, **W, x;                         A,B,W are K×5 matrices
{
       int i;
       double y;

       y = x;                                    initial input to first SOS4

       for (i=0; i<K; i++)
              y = sos4(A[i], B[i], W[i], y);     output of ith section

       return y;                                 final output from last SOS4
}
```

The $K\times 5$ matrices A and B are generated by the filter design MATLAB functions, such as `bpsbutt.m`, `bpcheb2.m`, or `bscheb2.m`, and can be passed as inputs to the above C functions. This can be done by saving the A and B outputs of the MATLAB functions into ASCII files and then invoking the following filtering routine `casfilt4.c` which is analogous to `casfilt.c` of Problem 7.15 and which reads A and B dynamically from the command line:

```
/* casfilt4.c - IIR filtering in cascade of fourth-order sections */

#include <stdlib.h>
#include <stdio.h>

#define MAX 64                                   initial allocation size

double cas4();

void main(int argc, char **argv)
{
    FILE *fpa, *fpb;                             coefficient files
    double **A, **B, **W, x, y;
    double *a;                                   temporary coefficient array
    int M, K, i, j, m;
    int max = MAX, dmax = MAX;                   allocation and increment

    if (argc != 3) {
        fprintf(stderr, "Usage: casfilt4 A.dat B.dat <x.dat >y.dat\n");
        exit(0);
        }

    if ((fpa = fopen(argv[1], "r")) == NULL) {
        fprintf(stderr, "Can't open filter file: %s\n", argv[1]);
        exit(0);
        }

    if ((fpb = fopen(argv[2], "r")) == NULL) {
        fprintf(stderr, "Can't open filter file: %s\n", argv[2]);
        exit(0);
        }

    a = (double *) calloc(max + 1, sizeof(double));
```

```
    for (M=0;; M++) {
        if (M == max) {
            max += dmax;
            a = (double *) realloc((char *) a, (max + 1) * sizeof(double));
            }
        if (fscanf(fpa, "%lf", a + M) == EOF) break;
        }
    a = (double *) realloc((char *) a, (M + 1) * sizeof(double));

    if (M%5 != 0) {
        fprintf(stderr, "all rows of A must be 5-dimensional");
        exit(0);
        }
    else
       K = M / 5;                                          number of SOS4

    A    = (double **) calloc(K, sizeof(double *));       allocate K rows
    B    = (double **) calloc(K, sizeof(double *));
    W    = (double **) calloc(K, sizeof(double *));       states of each SOS4
    for (i=0; i<K; i++) {
       A[i] = (double *) calloc(5, sizeof(double));       each row is 5-dim
       B[i] = (double *) calloc(5, sizeof(double));
       W[i] = (double *) calloc(5, sizeof(double));
       }

    for(i=0; i<K; i++)                                     form A matrix
       for(j=0; j<5; j++)
           A[i][j] = a[5*i + j];

    for(i=0; i<K; i++)                                     read B matrix
       for(j=0; j<5; j++)
           fscanf(fpb, "%lf", B[i] + j);

    while(scanf("%lf", &x) != EOF) {                       process input samples
        y = cas4(K, A, B, W, x);
        printf("%lf\n", y);
        }
}
```

The MATLAB version of `sos4` is as follows:

```
% sos4.m - fourth order section
%
% [y, w] = sos4(b, a, w, x)
%
% b = [b0, b1, b2, b3, b4] = 5-dim numerator (row vector)
% a = [1,  a1, a2, a3, a4] = 5-dim denominator (row vector)
% w = [w0, w1, w2, w3, w4] = 5-dim filter state (row vector)
% x = scalar input
% y = scalar output

function [y, w] = sos4(b, a, w, x)

w(1) = x - a(2:5) * w(2:5)';
y = b * w';
w = delay(4, w);
```

The MATLAB version of `cas4`:


```
% cas4.m - cascade of fourth order sections
%
% [y, W] = cas4(K, B, A, W, x)
%
% B = Kx5 numerator matrix
% A = Kx5 denominator matrix
% W = Kx5 state matrix
% x = scalar input
% y = scalar output

function [y, W] = cas4(K, B, A, W, x)

y = x;

for i = 1:K,
       [y, W(i,:)] = sos4(B(i,:), A(i,:), W(i,:), y);
end
```

The MATLAB version of `casfilt4` for filtering of a long input vector is as follows. It is analogous to `casfilt.m` of Problem 7.15:

```
% casfilt4.m - cascade of fourth-order sections
%
% y = casfilt4(B, A, x)
%
% B = Kx5 numerator matrix
% A = Kx5 denominator matrix
% x = row vector input
% y = row vector output

function y = casfilt4(B, A, x)

[K, K1] = size(A);
[N1, N] = size(x);
W = zeros(K,5);

for n = 1:N,
       [y(n), W] = cas4(K, B, A, W, x(n));
end
```

Problem 11.22

The following MATLAB code designs the pairs of lowpass, highpass, bandpass, and bandstop Butterworth filters, and shows how to compute the phase response and impulse response of one of the filters (the first bandpass filter):

```
Apass = -10*log10(0.98);                                   stringent case
Astop = -10*log10(0.02);

[A1, B1, P1] = lhbutt(1, 20, 4, 5, 0.5, 10);               lowpass
[A2, B2, P2] = lhbutt(1, 20, 4, 5, Apass, Astop);          lowpass

[A3, B3, P3] = lhbutt(-1, 20, 5, 4, 0.5, 10);              highpass
[A4, B4, P4] = lhbutt(-1, 20, 5, 4, Apass, Astop);         highpass

[A5, B5, P5] = bpsbutt(1, 20, 2, 4, 1.5, 4.5, 0.5, 10);    bandpass
[A6, B6, P6] = bpsbutt(1, 20, 2, 4, 1.5, 4.5, Apass, Astop); bandpass
```

```
[A7, B7, P7] = bpsbutt(-1, 20, 1.5, 4.5, 2, 4, 0.5, 10);          bandstop
[A8, B8, P8] = bpsbutt(-1, 20, 1.5, 4.5, 2, 4, Apass, Astop);     bandstop

om = (0:199) * pi / 200;                                  200 frequencies in [0, π]

a5 = cas2can(A5);                                         bandpass case
b5 = cas2can(B5);
H5 = dtft(b5, om)./dtft(a5, om);                          frequency response
argH5 = angle(H5) * 180 / pi;                             phase response (in degrees)

x  = [1, zeros(1, 99)];                                   length-100 unit impulse
h5 = casfilt4(B5, A5, x);                                 impulse response
y5 = filter(b5, a5, x);                                   conventional method. Note, y5=h5
```

The computed phase and impulse responses are shown in Fig. P11.3. Notice how the phase response is almost linear within its passband.

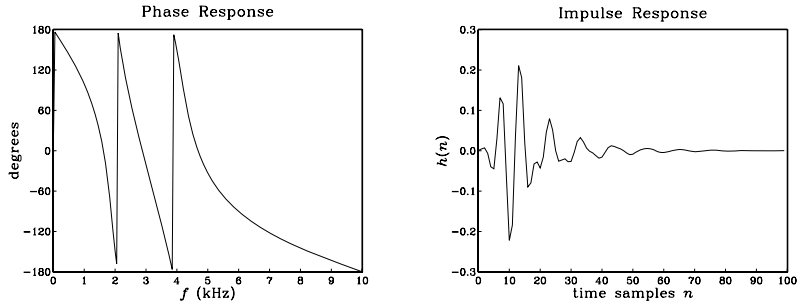


Fig. P11.3 Phase and impulse responses of (the less stringent) bandpass filter of Problem 11.22.

Problem 11.23

The following MATLAB code designs the pairs of lowpass, highpass, bandpass, and bandstop Chebyshev filters, and shows how to compute the phase response and impulse response of one of the filters (the first bandpass filter):

```
Apass = -10*log10(0.98);                                  stringent case
Astop = -10*log10(0.02);

[A1, B1, P1] = lhcheb1(1, 20, 4, 5, 0.5, 10);             LP type 1
[A2, B2, P2] = lhcheb1(1, 20, 4, 5, Apass, Astop);        LP type 1

[A3, B3, P3] = lhcheb1(-1, 20, 5, 4, 0.5, 10);            HP type 1
[A4, B4, P4] = lhcheb1(-1, 20, 5, 4, Apass, Astop);       HP type 1

[A5, B5, P5] = lhcheb2(1, 20, 4, 5, 0.5, 10);             LP type 2
[A6, B6, P6] = lhcheb2(1, 20, 4, 5, Apass, Astop);        LP type 2

[A7, B7, P7] = bpcheb2(20, 2, 4, 1.5, 4.5, 0.5, 10);      BP type 2
[A8, B8, P8] = bpcheb2(20, 2, 4, 1.5, 4.5, Apass, Astop); BP type 2
```



```
[A9, B9, P9]    = bscheb2(20, 1.5, 4.5, 2, 4, 0.5, 10);   BS type 2
[A10, B10, P10] = bscheb2(20, 1.5, 4.5, 2, 4, Apass, Astop); BS type 2

om = (0:199) * pi / 200;                                  200 frequencies in [0, π]

a7 = cas2can(A7);                                         bandpass case
b7 = cas2can(B7);
H7 = dtft(b7, om) ./ dtft(a7, om);                        frequency response
argH7 = angle(H7) * 180 / pi;                             phase response (in degrees)

x  = [1, zeros(1, 99)];                                   length-100 unit impulse
h7 = casfilt4(B7, A7, x);                                 impulse response
y7 = filter(b7, a7, x);                                   conventional method. Note, y7=h7
```

The computed phase and impulse responses are shown in Fig. P11.4. Note that the Chebyshev phase response is not as linear within its passband.

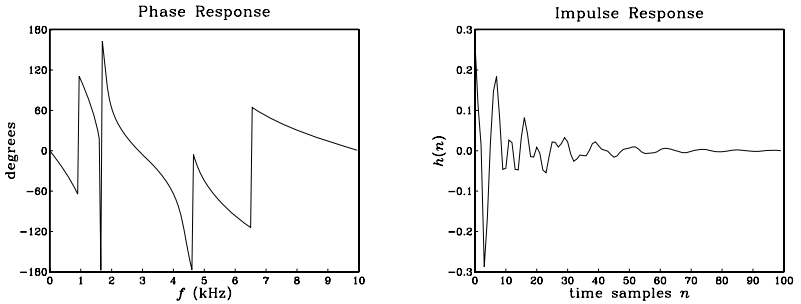


Fig. P11.4 Phase and impulse responses of (the less stringent) bandpass filter of Problem 11.23.

Chapter 12 Problems

Problem 12.1

The low-rate transfer functions, $D_i(z)$ and $H_i(z) = z^{-2}D_i(z)$, corresponding to Eq. (12.4.2) will be:

$$\begin{aligned}
D_0(z) &= 1 \\
D_1(z) &= -0.13z^2 + 0.30z + 0.90 - 0.18z^{-1} \\
D_2(z) &= -0.21z^2 + 0.64z + 0.64 - 0.21z^{-1} \\
D_3(z) &= -0.18z^2 + 0.90z + 0.30 - 0.13z^{-1} \\
H_0(z) &= z^{-2} \\
H_1(z) &= -0.13 + 0.30z^{-1} + 0.90z^{-2} - 0.18z^{-3} \\
H_2(z) &= -0.21 + 0.64z^{-1} + 0.64z^{-2} - 0.21z^{-3} \\
H_3(z) &= -0.18 + 0.90z^{-1} + 0.30z^{-2} - 0.13z^{-3}
\end{aligned}$$

Setting $z = \zeta^4$ and using Eq. (12.2.15), we obtain the transfer function $D(\zeta)$, which is recognized as the double-sided ζ-transform of the sequence **d** given in Eq. (12.4.1):

$$\begin{aligned}
D(\zeta) &= D_0(\zeta^4) + \zeta^{-1}D_1(\zeta^4) + \zeta^{-2}D_2(\zeta^4) + \zeta^{-3}D_3(\zeta^4) \\
&= 1 + \zeta^{-1}(-0.13\zeta^8 + 0.30\zeta^4 + 0.90 - 0.18\zeta^{-4}) \\
&\quad + \zeta^{-2}(-0.21\zeta^8 + 0.64\zeta^4 + 0.64 - 0.21\zeta^{-4}) \\
&\quad + \zeta^{-3}(-0.18\zeta^8 + 0.90\zeta^4 + 0.30 - 0.13\zeta^{-4}) \\
&= 1 + 0.90(\zeta + \zeta^{-1}) + 0.64(\zeta^2 + \zeta^{-2}) + 0.30(\zeta^3 + \zeta^{-3}) \\
&\quad - 0.18(\zeta^5 + \zeta^{-5}) - 0.21(\zeta^6 + \zeta^{-6}) - 0.13(\zeta^7 + \zeta^{-7})
\end{aligned}$$

Problem 12.2

In this case, $L = 2$ and we choose $M = 2$, so that $N = 2LM + 1 = 9$. The ideal impulse response is:

$$d(k') = \frac{\sin(\pi k'/2)}{\pi k'/2}, \qquad -4 \le k' \le 4$$

or, numerically,

$$\mathbf{h} = \mathbf{d} = [0, -0.21, 0, 0.64, 1, 0.64, 0, -0.21, 0\ ]$$

It is depicted in Fig. P12.1.
The two polyphase subfilters are defined by Eq. (12.2.10), that is, for $i = 0, 1$,

$$d_i(k) = d(2k + i), \qquad -2 \le k \le 1$$

They can be extracted from **h** by taking every second entry, starting with the ith entry:

$$\mathbf{h}_0 = \mathbf{d}_0 = [0,\ 0,\ 1,\ 0]$$

$$\mathbf{h}_1 = \mathbf{d}_1 = [-0.21,\ 0.64,\ 0.64,\ -0.21]$$

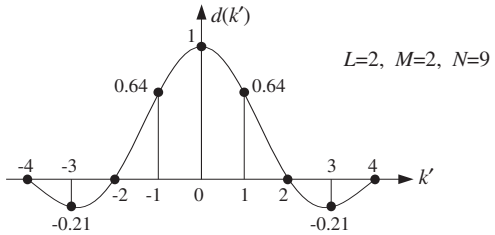


Fig. P12.1 Length-9 symmetric impulse response of 2-fold FIR interpolator.

The interpolated samples between $x(n) = x_{\rm up}(2n)$ and $x(n+1) = x_{\rm up}(2n+2)$ are calculated from Eqs. (12.2.18). The two subfilters act on the time-advanced (by $M = 2$) low-rate input samples $\{x(n+2),\ x(n+1),\ x(n),\ x(n-1)\}$, or, $\{x_{\rm up}(2n+4),\ x_{\rm up}(2n+2),\ x_{\rm up}(2n),\ x_{\rm up}(2n-2)\}$. Equations (12.2.18) can be cast in a compact matrix form:

$$\begin{bmatrix} y_{\rm up}(2n) \\ y_{\rm up}(2n+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -0.21 & 0.64 & 0.64 & -0.21 \end{bmatrix} \begin{bmatrix} x_{\rm up}(2n+4) \\ x_{\rm up}(2n+2) \\ x_{\rm up}(2n) \\ x_{\rm up}(2n-2) \end{bmatrix}$$

These filtering equations can also be obtained by superimposing the symmetric impulse response of Fig. P12.1 on each of the contributing low-rate samples

$$\{x_{\rm up}(2n+4),\ x_{\rm up}(2n+2),\ x_{\rm up}(2n),\ x_{\rm up}(2n-2)\}$$

and adding up their contributions at each of the intermediate sampling instants $2n + i$, $i = 0, 1$, as shown in Fig. P12.2.

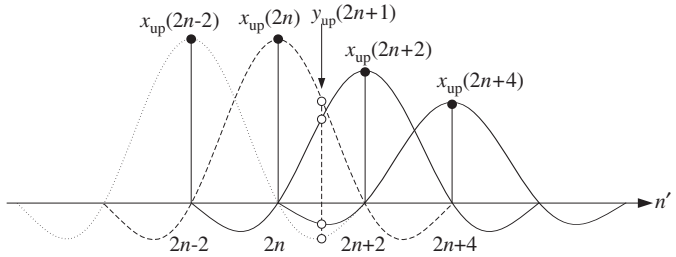


Fig. P12.2 Superposition of impulse responses.

The Hamming windowed version of the filter is obtained by multiplying the full length-9 filter response **h** by a length-9 Hamming window. The resulting impulse response will be:

$$\mathbf{h} = [0, -0.05, 0, 0.55, 1, 0.55, 0, -0.05, 0\ ]$$

The corresponding polyphase subfilters are obtained by extracting every second entry:

$$\mathbf{h}_0 = [0,\ 0,\ 1,\ 0]$$

$$\mathbf{h}_1 = [-0.05,\ 0.55,\ 0.55,\ -0.05]$$

The interpolation equations will be in this case:

$$\begin{bmatrix} y_{\rm up}(2n) \\ y_{\rm up}(2n+1) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -0.05 & 0.55 & 0.55 & -0.05 \end{bmatrix} \begin{bmatrix} x_{\rm up}(2n+4) \\ x_{\rm up}(2n+2) \\ x_{\rm up}(2n) \\ x_{\rm up}(2n-2) \end{bmatrix}$$

Problem 12.3

For the case of a 3-fold interpolator, we have $L = 3$ and choose again $M = 2$, corresponding to filter length $N = 2LM + 1 = 13$. The ideal impulse response is:

$$d(k') = \frac{\sin(\pi k'/3)}{\pi k'/3}, \qquad -6 \le k' \le 6$$

or, numerically,

$$\mathbf{h} = \mathbf{d} = [0, -0.17, -0.21, 0, 0.41, 0.83, 1, 0.83, 0.41, 0, -0.21, -0.17, 0]$$

where **h** is the causal version, with time origin at the left of the vector, and **d** is the symmetric one, with time origin at the middle of the vector. This impulse response is shown in Fig. P12.3. The three polyphase subfilters are defined by Eq. (12.2.10), that is, for $i = 0, 1, 2$,

$$d_i(k) = d(3k + i), \qquad -2 \le k \le 1$$

They can be extracted from **h** by taking every third entry, starting with the ith entry:

$$\mathbf{h}_0 = \mathbf{d}_0 = [0,\ 0,\ 1,\ 0]$$

$$\mathbf{h}_1 = \mathbf{d}_1 = [-0.17,\ 0.41,\ 0.83,\ -0.21]$$

$$\mathbf{h}_2 = \mathbf{d}_2 = [-0.21,\ 0.83,\ 0.41,\ -0.17]$$

The interpolated samples between $x(n) = x_{\rm up}(3n)$ and $x(n+1) = x_{\rm up}(3n+3)$ are calculated from Eqs. (12.2.18).

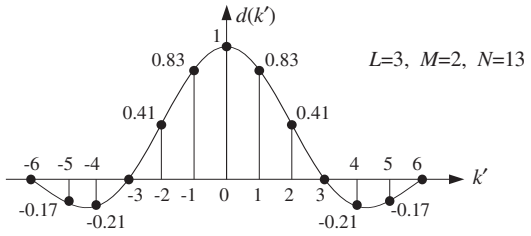


Fig. P12.3 Length-13 symmetric impulse response of 3-fold FIR interpolator.

All three subfilters act on the time-advanced (by $M = 2$) low-rate input samples $\{x(n+2), x(n+1), x(n), x(n-1)\}$, or, $\{x_{\rm up}(3n+6), x_{\rm up}(3n+3), x_{\rm up}(3n), x_{\rm up}(3n-3)\}$. Equations (12.2.18) can be cast in a compact matrix form:

$$\begin{bmatrix} y_{\rm up}(3n) \\ y_{\rm up}(3n+1) \\ y_{\rm up}(3n+2) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -0.17 & 0.41 & 0.83 & -0.21 \\ -0.21 & 0.83 & 0.41 & -0.17 \end{bmatrix} \begin{bmatrix} x_{\rm up}(3n+6) \\ x_{\rm up}(3n+3) \\ x_{\rm up}(3n) \\ x_{\rm up}(3n-3) \end{bmatrix}$$

These filtering equations can also be obtained by superimposing the symmetric impulse response of Fig. P12.3 on each of the contributing low-rate samples

$$\{x_{\rm up}(3n+6),\ x_{\rm up}(3n+3),\ x_{\rm up}(3n),\ x_{\rm up}(3n-3)\}$$

and adding up their contributions at each of the intermediate sampling instants $3n+i$, $i = 0, 1, 2$, as shown in Fig. P12.4.

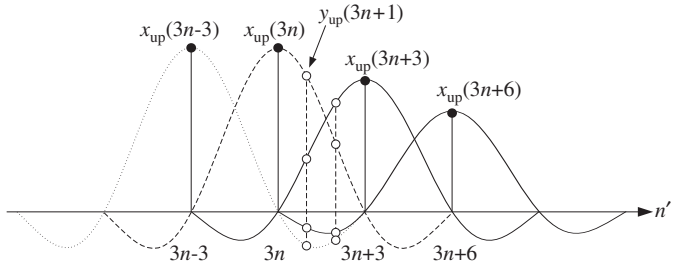


Fig. P12.4 Superposition of impulse responses.

The Hamming windowed version of the filter is obtained by multiplying the full length-13 filter response **h** by a length-13 Hamming window. The resulting impulse response will be:

$$\mathbf{h} = [0, -0.02, -0.06, 0, 0.32, 0.78, 1, 0.78, 0.32, 0, -0.06, 0.02, 0]$$

The corresponding polyphase subfilters are obtained by extracting every third entry:

$$\mathbf{h}_0 = [0,\ 0,\ 1,\ 0]$$

$$\mathbf{h}_1 = [-0.02,\ 0.32,\ 0.78,\ -0.06]$$

$$\mathbf{h}_2 = [-0.06,\ 0.78,\ 0.32,\ -0.02]$$

The interpolation equations will be in this case:

$$\begin{bmatrix} y_{\rm up}(3n) \\ y_{\rm up}(3n+1) \\ y_{\rm up}(3n+2) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ -0.02 & 0.32 & 0.78 & -0.06 \\ -0.06 & 0.78 & 0.32 & -0.02 \end{bmatrix} \begin{bmatrix} x_{\rm up}(3n+6) \\ x_{\rm up}(3n+3) \\ x_{\rm up}(3n) \\ x_{\rm up}(3n-3) \end{bmatrix}$$

Problem 12.4

For interpolation factor $L = 4$ and filter length $N = 25$, we find:

$$N = 2LM + 1 \qquad \Rightarrow \qquad M = (N-1)/2L = (25-1)/8 = 3$$

Thus, we use 3 low-rate samples above and three below every interpolated value to be computed. The length-25 ideal interpolation impulse response is calculated from:

$$d(k') = \frac{\sin(\pi k'/4)}{\pi k'/4}, \qquad \text{for} \quad -12 \le k' \le 12$$

This generates the numerical values:

$$\mathbf{d} = [\ 0.000,\ \ 0.082,\ \ 0.127,\ \ 0.100,\ \ 0.000,\ -0.129,\ -0.212,\ -0.180,$$
$$0.000,\ \ 0.300,\ \ 0.637,\ \ 0.900,\ \ 1.000,\ \ 0.900,\ \ 0.637,\ \ 0.300,$$
$$0.000,\ -0.180,\ -0.212,\ -0.129,\ \ 0.000,\ \ 0.100,\ \ 0.127,\ \ 0.082,\ \ 0.000\ ]$$

This impulse response is shown in Fig. P12.5 (where the values have been rounded to 2-digit accuracy for convenience.)

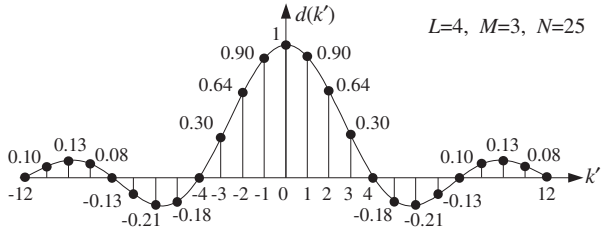


Fig. P12.5 Length-25 symmetric impulse response of 4-fold FIR interpolator.

Given $2M = 6$ low-rate samples $\{A, B, C, D, E, F\}$ as shown in Fig. 12.1.11, the three values $\{X, Y, Z\}$ interpolated between C and D are calculated by convolving $\mathbf{d}$ with the upsampled low-rate samples. Using the flip-and-slide form of convolution as in Fig. 12.4.3, we position the impulse response $\mathbf{d}$ at the three successive positions, read off the values of $d(k')$ where it intersects with the low-rate samples, and add up the results. This procedure is shown in Fig. P12.6. The corresponding linear combinations of low-rate samples are precisely those of Eq. (12.1.3). They may be rewritten in the polyphase matrix form (with the low-rate samples listed from the latest down to the earliest):

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 0.082 & -0.129 & 0.300 & 0.900 & -0.180 & 0.100 \\ 0.127 & -0.212 & 0.637 & 0.637 & -0.212 & 0.127 \\ 0.100 & -0.180 & 0.900 & 0.300 & -0.129 & 0.082 \end{bmatrix} \begin{bmatrix} F \\ E \\ D \\ C \\ B \\ A \end{bmatrix}$$

Problem 12.5

The ideal 3-fold interpolation case with $M = 2$, so that $N = 2LM + 1 = 2 \cdot 3 \cdot 2 + 1 = 13$, was discussed in Problem 12.3. For the linear interpolator, using Eq. (12.3.7), we have:

$$\begin{bmatrix} y_{\text{up}}(3n) \\ y_{\text{up}}(3n+1) \\ y_{\text{up}}(3n+2) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 1/3 & 2/3 & 0 \\ 0 & 2/3 & 1/3 & 0 \end{bmatrix} \begin{bmatrix} x(n+2) \\ x(n+1) \\ x(n) \\ x(n-1) \end{bmatrix} = \begin{bmatrix} 1/3 & 2/3 \\ 2/3 & 1/3 \end{bmatrix} \begin{bmatrix} x(n+1) \\ x(n) \end{bmatrix}$$

Problem 12.6

The following complete C program `interp.c` implements the polyphase sample-by-sample processing algorithm of Eq. (12.2.20) and the initialization procedure of (12.2.19):

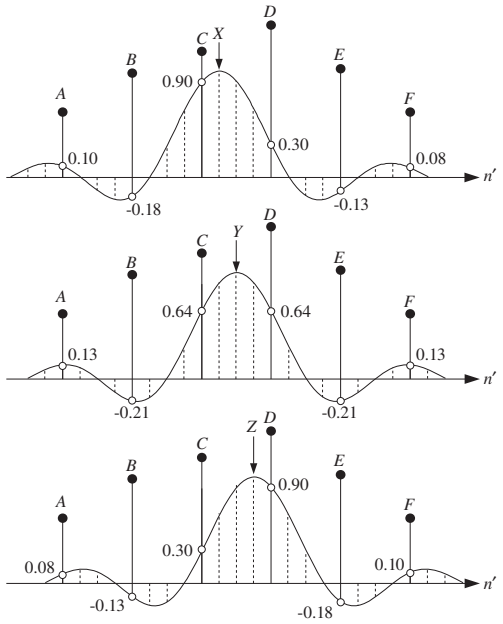


Fig. P12.6 Interpolated values are calculated by the flip-and-slide form of convolution.

```
/* interp.c - L-fold FIR interpolator (rectangular and Hamming window design)
 *
 * Usage:  interp hamm L M < xlow.dat > yhigh.dat
 *
 * hamm      = 0/1 for rectangular/Hamming window
 * L         = interpolation factor
 * M         = half-length of polyphase subfilters (length = 2M, order = P = 2M-
1)
 * xlow.dat  = file containing low-rate input samples
 * yhigh.dat = file containing high-rate interpolated output samples
*/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

void delay();
double dot();
double d();                                          ideal interpolation filter

void main(int argc, char ** argv)
{
    int i, n, L, M, P, hamm;
    double x, *y, *w, **h, pi = 4 * atan(1.0);

    if (argc != 4) {
```

```
        fprintf(stderr,"\nUsage: interp hamm L M < xlow.dat > yhigh.dat\n\n");
        fprintf(stderr,"where: hamm      = 0/1 for rectangular/hamm window\n");
        fprintf(stderr,"       L         = interpolation factor\n");
        fprintf(stderr,"       M         = half-length of polyphase subfilters\n");
        fprintf(stderr,"       xlow.dat  = file for low-rate input samples\n");
        fprintf(stderr,"       yhigh.dat = file for high-rate interpolated output samples\n");
        exit(0);
        }

    hamm = atoi(argv[1]);
    L    = atoi(argv[2]);
    M    = atoi(argv[3]);

    P = 2*M - 1;                                          polyphase filter order

    h = (double **) calloc(L, sizeof(double *));          allocate L subfilters
    for (i=0; i<L; i++)
        h[i] = (double *) calloc(2*M, sizeof(double));    each of length 2M

    y = (double *) calloc(L, sizeof(double));             allocate L polyphase outfprintf(stderr,

    w = (double *) calloc(2*M, sizeof(double));           allocate low-rate internal state

    for (i=0; i<L; i++)                                   construct polyphase subfilters
        for (n=0; n<2*M; n++)
           if (hamm == 0)                                 rectangular window
              h[i][n] = d(L, L*n+i-L*M);
           else                                           Hamming window
              h[i][n] = d(L, L*n+i-L*M) * (0.54 - 0.46*cos((L*n+i)*pi/(L*M)));


                                                          initialize by Eq. (12.2.19)
    for (i=M; i>=1; i--)                                  initialize state w by reading
         scanf("%lf", w + i);                             first M low-rate input samples


                                                          process by Eq. (12.2.20)
    while(scanf("%lf", &x) != EOF) {                      keep reading low-rate samples till EOF
        w[0] = x;
        for (i=0; i<L; i++) {
           y[i] = dot(P, h[i], w);                        ith interpolated output sample
           printf("%.12lf\n", y[i]);                      and write it into stdout
           }
        delay(P, w);                                      update low-rate delay line
        }

    for (n=0; n<M; n++) {                                 input-off transients
        w[0] = 0.0;                                       zero input
        for (i=0; i<L; i++) {
           y[i] = dot(P, h[i], w);                        ith interpolated output sample
           printf("%.12lf\n", y[i]);                      and write it into stdout
           }
        delay(P, w);                                      update low-rate delay line
        }
}

/* ------------------------------------------------------- */

/* d(L, k) - ideal L-fold interpolation coefficients */

double d(L, k)
int L, k;
{
    double t, pi = 4 * atan(1.0);

    if (k == 0)                                                    d(0) = 1
        return (1.0);
    else {
        if (k%L == 0)                                  d(k) vanishes at non-zero multiples of L
            return (0.0);
        else {
            t = pi * k / L;
            return (sin(t)/t);                         typical sin x/x impulse response
            }
         }
}
```

The program has usage:

```
interp hamm L M < xlow.dat > yhigh.dat
```

where the parameter `hamm` specifies a Hamming or rectangular window, L is the interpolation factor, and M is the half-length of the polyphase subfilters, so that the total filter length is $N = 2LM + 1$.

The program designs the windowed length-N interpolation filter and then reads the low-rate input samples sequentially from a file `xlow.dat` or `stdin`, computes the high-rate interpolated samples, and writes them to the file `yhigh.dat` or to the `stdout`. Each low-rate input generates L high-rate outputs. Inside the program, the rectangular and Hamming window design of the impulse response may be replaced easily by a Kaiser design.

The program `interp.c` uses a linear-buffer version of the low-rate delay line used by all the polyphase subfilters. The following program `cinterp.c` is the circular-buffer version of `interp.c`:

```
/* cinterp.c - circular-buffer version of interp.c */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

void cdelay(), wrap();
double cdot();                                            circular version of dot.c
double d();                                               ideal interpolation filter

void main(int argc, char ** argv)
{
    int i, n, L, M, P, hamm;
    double x, *y, *w, *p, **h, pi = 4 * atan(1.0);

    if (argc != 4) {
        fprintf(stderr,"\nUsage: cinterp hamm L M < xlow.dat > yhigh.dat\n\n");
        fprintf(stderr,"where: hamm      = 0/1 for rectangular/hamm window\n");
        fprintf(stderr,"       L         = interpolation factor\n");
        fprintf(stderr,"       M         = half-length of polyphase subfilters\n");
        fprintf(stderr,"       xlow.dat  = file for low-rate input samples\n");
```

```
        fprintf(stderr,"       yhigh.dat = file for high-rate interpolated output samples\n");
        exit(0);
        }

    hamm = atoi(argv[1]);
    L    = atoi(argv[2]);
    M    = atoi(argv[3]);

    P = 2*M - 1;                                              polyphase filter order

    h = (double **) calloc(L, sizeof(double *));              allocate L subfilters
    for (i=0; i<L; i++)
        h[i] = (double *) calloc(2*M, sizeof(double));        each of length 2M

    y = (double *) calloc(L, sizeof(double));                 allocate L polyphase outfprintf(stderr,

    w = (double *) calloc(2*M, sizeof(double));               allocate low-rate internal state
    p = w;                                                    initialize circular pointer

    for (i=0; i<L; i++)                                       construct polyphase subfilters
        for (n=0; n<2*M; n++)
           if (hamm == 0)                                     rectangular window
              h[i][n] = d(L, L*n+i-L*M);
           else                                               Hamming window
              h[i][n] = d(L, L*n+i-L*M) * (0.54 - 0.46*cos((L*n+i)*pi/(L*M)));


                                                              initialize by Eq. (P12.1)
    for (i=M; i>=1; i--) {                                    initialize state w by reading
         scanf("%lf", p);                                     first M low-rate input samples
         cdelay(P, w, &p);
         }


                                                              process by Eq. (P12.2)
    while(scanf("%lf", &x) != EOF) {                          keep reading low-rate samples till EOF
        *p = x;                                               beginning of state vector
        for (i=0; i<L; i++) {
           y[i] = cdot(P, h[i], w, p);                        circular version of dot()
           printf("%.12lf\n", y[i]);                          and write it into stdout
           }
        cdelay(P, w, &p);                                     update low-rate delay line
        }

    for (n=0; n<M; n++) {                                     input-off transients
        *p = 0.0;                                             zero input
        for (i=0; i<L; i++) {
           y[i] = cdot(P, h[i], w, p);                        circular version of dot()
           printf("%.12lf\n", y[i]);                          and write it into stdout
           }
        cdelay(P, w, &p);                                     update low-rate delay line
        }
}

/* ------------------------------------------------------- */

/* d(L, k) - ideal L-fold interpolation coefficients */

double d(L, k)
int L, k;
{
    double t, pi = 4 * atan(1.0);

    if (k == 0)                                                                  d(0) = 1
        return (1.0);
    else {
        if (k%L == 0)                          d(k) vanishes at non-zero multiples of L
            return (0.0);
        else {
            t = pi * k / L;
            return (sin(t)/t);                 typical sin x/x impulse response
            }
         }
}

/* ------------------------------------------------------- */

/* cdot(P, h, w, p) - circular version of dot.c */

double tap();

double cdot(P, h, w, p)                                       computes dot product of h with
double *h, *w, *p;                                            current internal state vector
int P;                                                        pointed to by pointer p
{
       int i;
       double y;

       for (y=0, i=0; i<=P; i++)
              y += h[i] * tap(P, w, p, i);

       return y;
}
```


It uses a circular pointer p that circulates over the low-rate buffer $\mathbf{w}$. The circular-buffer version of Eqs. (12.2.19) and (12.2.20), implemented by `cinterp.c`, can be stated as follows:

for $m = M$ down to $m = 1$ do:
    read low-rate input sample x
    $*p = x$
    $\text{cdelay}(P, \mathbf{w}, \&p)$

(P12.1)

and

for each low-rate input sample x do:
    $*p = x$
    for $m = 0, 1, \ldots, P$ determine internal states:
        $s_m = \text{tap}(P, \mathbf{w}, p, m)$
    for $i = 0, 1, \ldots, L-1$ compute output of ith filter:
        $y_i = \text{dot}(P, \mathbf{h}_i, \mathbf{s})$
    $\text{cdelay}(P, \mathbf{w}, \&p)$

(P12.2)

The length-17 rectangular and Hamming windowed filters for this problem were derived in Section 12.4.1. The given inputs were processed by `interp.c`. The upsampled versions of these inputs, $x_{\text{up}}(n')$, are shown in Fig. P12.7 together with the corresponding interpolated output signals $y_{\text{up}}(n')$, computed by the length-17 Hamming windowed version of the interpolation filter.

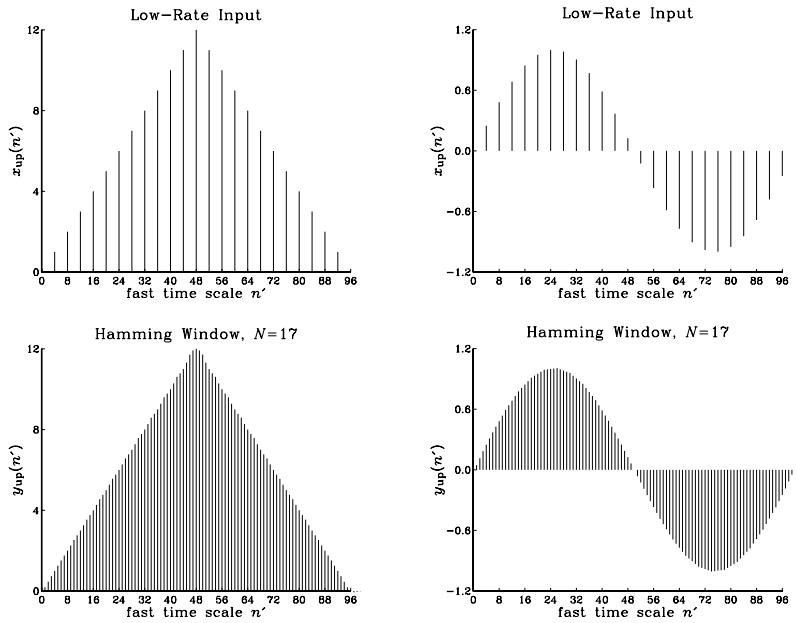


Fig. P12.7 Interpolation of triangular and sinusoidal signals.

Problem 12.7

We start with $f_s = 40$ kHz, $\Delta f = 5$ kHz, and $A = 80$ for all the stages. The effective single stage filter will have Kaiser length:

$$N - 1 = \frac{DL}{\Delta F} = 321.08$$

which is rounded up to:

$$N = 337 = 2LM + 1 = 16M + 1 \qquad \Rightarrow \qquad M = 21$$

The 2×4 multistage filters, shown in Fig. P12.8, have Kaiser lengths:

$$N_0 - 1 = \frac{L_0 D}{\Delta F} = 80.27, \qquad N_1 - 1 = \frac{DF_1}{F_0 - 1} = \frac{8D}{2 - 1} = 40.14$$

and are rounded up to:

$$N_0 = 85 = 2L_0M_0 + 1 = 4M_0 + 1 \qquad \Rightarrow \qquad M_0 = 21$$

$$N_1 = 49 = 2L_1M_1 + 1 = 8M_1 + 1 \qquad \Rightarrow \qquad M_1 = 6$$

The relative computational rate will be

$$\frac{R_{\text{multi}}}{R_{\text{single}}} = \frac{N_0 + N_1F_0}{N} = 0.54$$

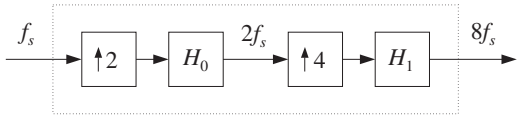


Fig. P12.8 2x4=8-fold oversampling filter.

which compares well with the approximation:

$$\frac{R_{\text{multi}}}{R_{\text{single}}} = \frac{1}{L_1} + \frac{L_0}{L_0 - 1}\Delta F = 0.50$$

Thus, the multistage realization requires only 54 percent the cost of the single-stage one. The combined filter $H_{\text{tot}}(f) = H_0(f)H_1(f)$ and the superimposed plots of the filters $H_0(f)$, $H_1(f)$ are shown in Fig. P12.9. Similarly, for the 4×2 case, shown in Fig. P12.10, we find the filter lengths:

$$N_0 = 169, \quad N_1 = 17$$

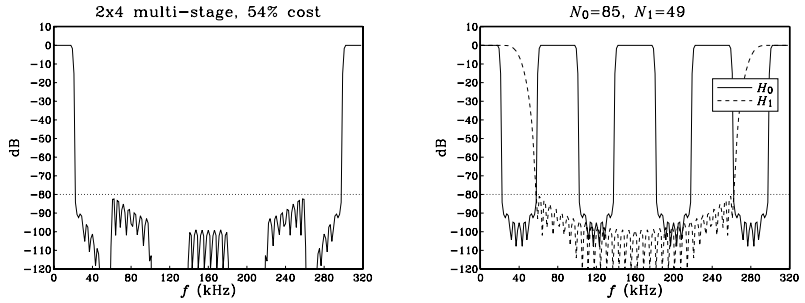


Fig. P12.9 Combined filter $H_0(f)H_1(f)$, and individual filters $H_0(f)$, $H_1(f)$.

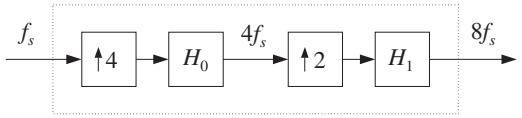


Fig. P12.10 4x2=8-fold oversampling filter.

The relative computational rate is in this case:

$$\frac{R_{\text{multi}}}{R_{\text{single}}} = \frac{N_0 + N_1F_0}{N} = 0.70$$

The combined filter $H_{\text{tot}}(f) = H_0(f)H_1(f)$ and the superimposed plots of the filters $H_0(f)$, $H_1(f)$ are shown in Fig. P12.11. Initially, we designed the filters with stopband attenuations $A_0 = A_1 = 80$ dB. However, we found that in order to keep the overall stopband attenuation

below 80 dB, the attenuation of the second filter had to be increased to $A_1 = 84$ dB. This did not change the filter lengths, but it changed slightly the Kaiser parameters D_1 and α_1 for that factor.

Finally, in the 2×2×2 case, shown in Fig. P12.12, we find the filter lengths

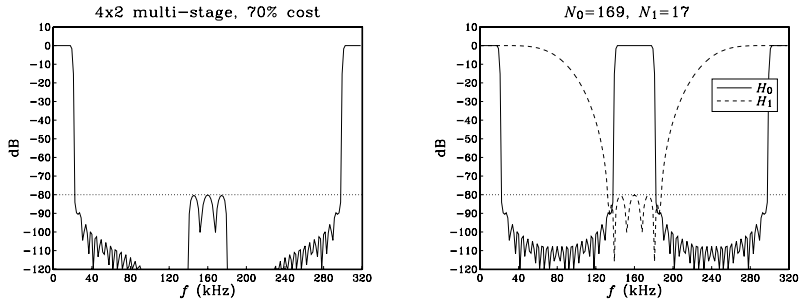


Fig. P12.11 Combined filter $H_0(f)H_1(f)$, and individual filters $H_0(f), H_1(f)$.

$$N_0 = 85, \quad N_1 = 25, \quad N_2 = 17$$

resulting in the relative computational rate

$$\frac{R_{\text{multi}}}{R_{\text{single}}} = \frac{N_0 + N_1F_0 + N_2F_1}{N} = 0.60$$

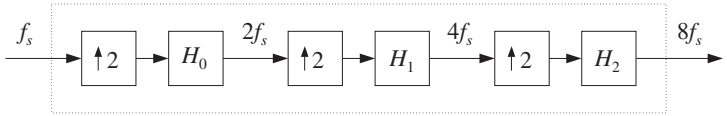


Fig. P12.12 2x2x2=8-fold oversampling filter.

The combined filter $H_{\text{tot}}(f) = H_0(f)H_1(f)H_2(f)$ and individual filters $H_0(f), H_1(f), H_2(f)$ are shown in Fig. P12.13. For reference, we also show in Fig. P12.14 the magnitude response of the single stage 8-fold oversampling filter. Again, the nominal attenuation of the last factor H_2 had to be increased to $A_2 = 84$ dB, whereas the first two were $A_0 = A_1 = 80$ dB.

Problem 12.8

In units of $f_s = 44.1$ kHz, the transition width from the edge of the passband to the edge of the stopband will be:

$$\Delta F = \frac{\Delta f}{f_s} = \frac{24.1 - 20}{44.1} = 0.0930$$

The Kaiser window width parameter is $D = (A - 7.95)/14.36 = (80 - 7.95)/14.36 = 5.0174$. Thus, for a single-stage design with oversampling ratio $L = 4$, we find the Kaiser length:

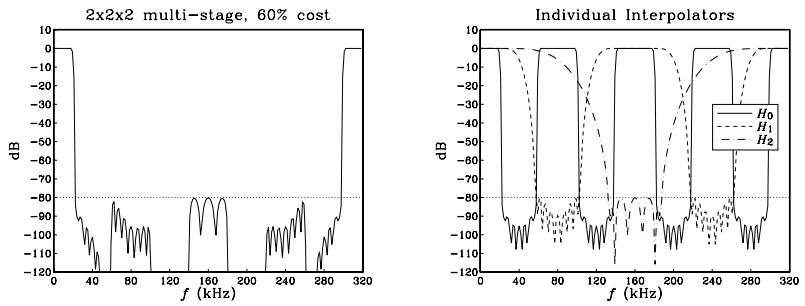


Fig. P12.13 Combined filter $H_0(f)H_1(f)H_2(f)$, and individual filters $H_0(f), H_1(f), H_2(f)$.

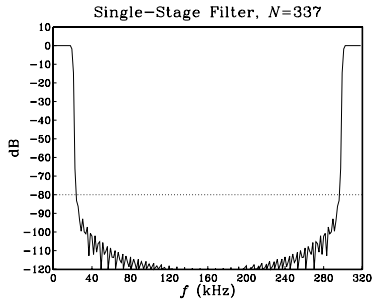


Fig. P12.14 Single-stage 8-fold oversampling filter.

$$N - 1 = \frac{DL}{\Delta F} = \frac{5.0174 \cdot 4}{0.0930} = 215.803 \qquad \Rightarrow \qquad N = 217, \quad M = (N-1)/(2L) = 27$$

The corresponding computational rate of the polyphase realization is

$$R_{\text{single}} = Nf_s = 217 \cdot 44.1 = 9569.7 \text{ MAC/msec} = 9.5697 \text{ MIPS}$$

(assuming that each MAC is done with one instruction.) Thus, a 20-MIPS DSP chip can easily handle it.

For the two-stage case, the design specifications are depicted in Fig. 12.2.9 of Section 12.2.5. Stage H_0 operating at $2f_s$ will have the narrow transition width $\Delta f = 4.1$ kHz, or $\Delta F = 0.0930$. This gives the Kaiser length:

$$N_0 - 1 = \frac{DL_0}{\Delta F} = \frac{5.0174 \cdot 2}{0.0930} = 107.9 \qquad \Rightarrow \qquad N_0 = 109, \quad M_0 = (N_0 - 1)/(2L_0) = 27$$

The second stage H_1 will have a wider transition width $\Delta f_1 = f_s$, or $\Delta F_1 = 1$. Thus, the filter length will be:

$$N_1 - 1 = \frac{D\Delta F_1}{F_0 - 1} = \frac{5.0174 \cdot 4}{1.000} = 20.07 \qquad \Rightarrow \qquad N_1 = 25, \quad M_1 = (N_1 - 1)/(2L_1) = 6$$

The polyphase computational rate of the two sections combined will be

$$R_{\text{multi}} = N_0 f_s + N_1 (2f_s) = 159 f_s = 159 \cdot 44.1 = 7011.9 \text{ MAC/msec} = 7.0119 \text{ MIPS}$$

Thus, the relative computational cost of the two-stage versus the single stage design will be:

$$\frac{R_{\text{multi}}}{R_{\text{single}}} = \frac{159 f_s}{217 f_s} = 0.73$$

Problem 12.9

The frequency responses can be computed by the MATLAB code:

```
f = (0:399) * 160 /400;                                  400 frequencies in [0,160] kHz

f1 = 28;                                                 normalization frequency for Butterworth
f2 = 16;                                                 normalization frequency for Bessel

H1 = 1 ./ (1 + 2*j*(f/f1) - 2*(f/f1).^2 - j*(f/f1).^3);          Butterworth
H2 = 15 ./ (15 + 15*j*(f/f2) - 6*(f/f2).^2 - j*(f/f2).^3);       Bessel

magH1 = 20 * log10(abs(H1));
magH2 = 20 * log10(abs(H2));

argH1 = angle(H1) * 180 / pi;
argH2 = angle(H2) * 180 / pi;
```

The phase and magnitude responses are shown in Figs. P12.15 and P12.16. The Bessel phase response is essentially linear over the 0–20 kHz passband. But its magnitude response is not as flat as the Butterworth's.

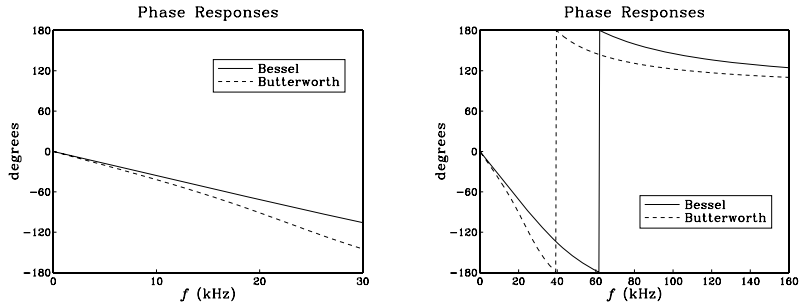


Fig. P12.15 Phase responses of Bessel and Butterworth filters.

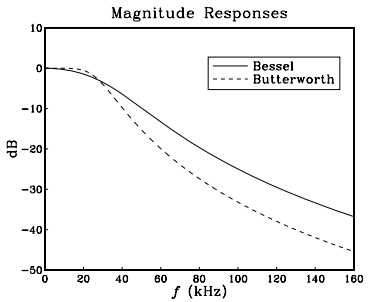


Fig. P12.16 Magnitude responses of Bessel and Butterworth filters.

Problem 12.10

The filter H_0 has a cutoff frequency $f_c = f_s/2$. The linear, hold, and DAC magnitude responses (normalized to 0 dB at DC) are in the general and the particular case of $L_0 = 4$, $L_1 = L_2 = 2$:

$$|H_1(f)| = \left| \frac{\sin(\pi f/L_0 f_s)}{L_1 \sin(\pi f/L_0 L_1 f_s)} \right|^2 = \left| \frac{\sin(\pi f/4f_s)}{2\sin(\pi f/8f_s)} \right|^2$$

$$|H_2(f)| = \left| \frac{\sin(\pi f/L_0 L_1 f_s)}{L_2 \sin(\pi f/L_0 L_1 L_2 f_s)} \right| = \left| \frac{\sin(\pi f/8f_s)}{2\sin(\pi f/16f_s)} \right|$$

$$|H_{\text{dac}}(f)| = \left| \frac{\sin(\pi f/L_0 L_1 L_2 f_s)}{\pi f/L_0 L_1 L_2 f_s} \right| = \left| \frac{\sin(\pi f/16f_s)}{\pi f/16f_s} \right|$$

The filter H_0 operates at rate $4f_s$ and cancels all the images at multiples of f_s which are not multiples of $4f_s$. The linear interpolator H_1 operates at rate $8f_s$ and vanishes at multiples of $4f_s$ which are not multiples of $8f_s$. The filter H_2 operates at rate $16f_s$ and vanishes at multiples of $8f_s$ which are not multiples of $16f_s$. Finally, H_{dac} vanishes at all non-zero multiples of $16f_s$. Thus, the combined effect of H_0, H_1, H_2, and H_{dac} is to (partially) remove all spectral images at multiples of f_s.

We note that the combined effect of the two hold transfer functions is an effective DAC operating at $8f_s$:

$$|H_2(f)H_{\text{dac}}(f)| = \left| \frac{\sin(\pi f/L_0 L_1 f_s)}{L_2 \sin(\pi f/L_0 L_1 L_2 f_s)} \right| \cdot \left| \frac{\sin(\pi f/L_0 L_1 L_2 f_s)}{\pi f/L_0 L_1 L_2 f_s} \right| = \left| \frac{\sin(\pi f/L_0 L_1 f_s)}{\pi f/L_0 L_1 f_s} \right| = \left| \frac{\sin(\pi f/8f_s)}{\pi f/8f_s} \right|$$

The reason for using H_2 is that the higher the L, the more bits can be saved via the noise-shaping requantizer.

Problem 12.11

The proof of the result may be seen graphically in Fig. P12.17 for the case $L = 4$. The ideal interpolator has cutoff $f_s/2$ and is periodic in f with period $Lf_s = 4f_s$. Thus, the passband replicates at multiples of $4f_s$.

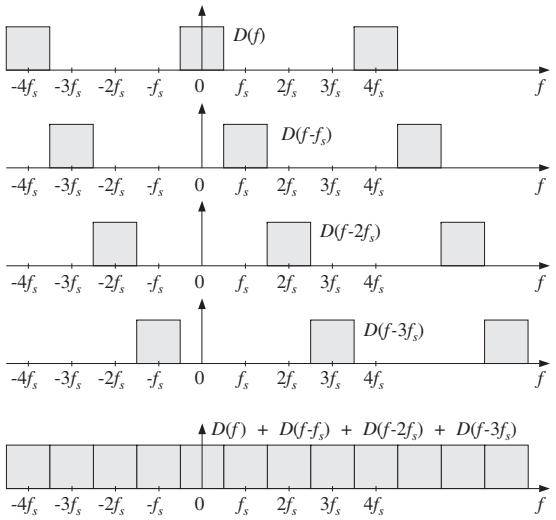


Fig. P12.17 Sum of successive shifts of ideal lowpass interpolator with $L = 4$ in Problem 12.11.

The terms $D(f - mf_s)$ correspond to the right shifts of $D(f)$ centered at mf_s. It is evident from the figure that the these shifted terms fill the gaps between the original replicas, resulting in a constant spectrum. The $1/L$ factor is needed to make the value of passband equal to unity. Thus, in the $L = 4$ case we have:

$$\frac{1}{4}\big[D(f)+D(f-f_s)+D(f-2f_s)+D(f-3f_s)\big] = 1, \quad \text{for all } f$$

Problem 12.12

Writing $k = Lk' + m$, $m = 0, 1, \ldots, L-1$, corresponds to the quotient and remainder of the division of k by L, and as such the integers $\{k', m\}$ are uniquely determined by k. Therefore, we may change the summation over k into a double summation over k' and m:

$$X(f) = \frac{1}{T}\sum_{k=-\infty}^{\infty} X_a(f - kf_s) = \frac{1}{T}\sum_{m=0}^{L-1}\sum_{k'=-\infty}^{\infty} X_a(f - (k'L+m)f_s)$$

$$= \frac{1}{T}\sum_{m=0}^{L-1}\sum_{k'=-\infty}^{\infty} X_a(f - (k'L+m)f_s) = \frac{1}{LT'}\sum_{m=0}^{L-1}\sum_{k'=-\infty}^{\infty} X_a(f - mf_s - k'f_s')$$

where we replaced $T = LT'$ and $Lf_s = f_s'$. If in the definition of $X'(f)$ we replace f by $f - mf_s$, we have:

$$X'(f - mf_s) = \frac{1}{T'}\sum_{k'=-\infty}^{\infty} X_a(f - mf_s - k'f_s')$$

Thus the k' summation in the above double sum may be replaced by $X'(f - mf_s)$ resulting in the desired result:

$$X(f) = \frac{1}{L}\sum_{m=0}^{L-1} X'(f - mf_s)$$

If we use the variables ω and ω', then $X(f)$ and $X'(f)$ are given by

$$X(\omega) = \sum_n x(n)e^{-j\omega n}, \qquad X'(\omega') = \sum_{n'} x'(n')e^{-j\omega' n'}$$

The shifted frequency $f - mf_s$ is in units of ω':

$$\frac{2\pi(f - mf_s)}{f_s'} = \frac{2\pi f}{f_s'} - \frac{2\pi m f_s}{Lf_s} = \omega' - \frac{2\pi m}{L}$$

Thus, using $\omega' = \omega/L$, we may write

$$X(\omega) = \frac{1}{L}\sum_{m=0}^{L-1} X'\Big(\omega' - \frac{2\pi m}{L}\Big) = \frac{1}{L}\sum_{m=0}^{L-1} X'(\omega - 2\pi m)/L)$$

Problem 12.13

The discrete-time sampling function $s'(n')$ consists of a periodic train of unit impulses at multiples of L, that is, at $n' = nL$, and separated by $L-1$ zeros between. Thus, the product of $s'(n')$ with $x'(n')$ will insert such zeros. But at the sampling times $n' = nL$ we will have $x_{\text{up}}(n') = x'(n') = x'(nL) = x(n)$.

Because of its periodicity in n' with period L, the sampling function admits a DFS expansion, that is, and inverse DFT expansion of the form of Eq. (9.7.1):

$$s'(n') = \frac{1}{L}\sum_{m=0}^{L-1} S'(\omega_m')e^{j\omega_m' n'}, \qquad \omega_m' = \frac{2\pi(mf_s)}{f_s'} = \frac{2\pi(mf_s'/L)}{f_s'} = \frac{2\pi m}{L} = m\text{th DFT frequency}$$

where $S'(\omega_m')$ is the L-point DFT of $s'(n')$ calculated from one period of $s'(n')$ as follows:

$$S'(\omega_m') = \sum_{n'=0}^{L-1} s'(n')e^{-j\omega_m' n'}$$

But within the first period $0 \le n' \le L-1$, the sampling function acts as a unit impulse, $s'(n') = \delta(n')$. Therefore, the n'-summation collapses to the term $n' = 0$, which is unity:

$$S'(\omega_m') = 1, \qquad m = 0, 1, \ldots, L-1$$

which gives for the inverse DFT:

$$s'(n') = \frac{1}{L}\sum_{m=0}^{L-1} e^{j\omega_m' n'} = \frac{1}{L}\sum_{m=0}^{L-1} e^{2\pi j m n'/L}$$

To transform the relationship $x_{\text{up}}(n') = s'(n')x'(n')$ to the frequency domain, we replace $s'(n')$ by its IDFT expression and use the modulation theorem of DTFTs to get:

$$x_{\text{up}}(n') = \frac{1}{L}\sum_{m=0}^{L-1} e^{j\omega_m' n'}x'(n') \qquad \Rightarrow \qquad X_{\text{up}}(\omega') = \frac{1}{L}\sum_{m=0}^{L-1} X'(\omega' - \omega_m')$$

But we have seen in Eq. (12.2.21) that the upsampled spectrum is the same as the original low-rate spectrum, that is, $X_{\text{up}}(f) = X(f)$, or in units of ω and ω', $X_{\text{up}}(\omega') = X(\omega)$. It follows that

$$X(\omega) = X_{\text{up}}(\omega') = \frac{1}{L}\sum_{m=0}^{L-1} X'(\omega' - \omega_m') = \frac{1}{L}\sum_{m=0}^{L-1} X'\Big(\omega' - \frac{2\pi m}{L}\Big)$$

Problem 12.14

Starting with the filtering equation $X'(f) = D(f)X(f)$, we form the sum of shifted high-rate spectra:

$$\frac{1}{L}\sum_{m=0}^{L-1} X'(f-mf_s) = \frac{1}{L}\sum_{m=0}^{L-1} D(f-mf_s)X(f-mf_s) = \frac{1}{L}\sum_{m=0}^{L-1} D(f-mf_s)X(f) = X(f)$$

where we used the periodicity of $X(f)$ with period f_s to write $X(f-mf_s) = X(f)$, and then we used the result of Problem 12.11 to replace the sum of shifted $D(f)$'s by unity.

Problem 12.15

The attenuation of an Nth order Butterworth filter in dB is given by

$$A(F) = 10\log_{10}\left[1 + \left(\frac{F}{F_0}\right)^{2N}\right]$$

where we used the normalized frequencies $F = f/f_s$ and $F_0 = f_0/f_s$. Given a required attenuation of A_{pass} dB over the passband $F_{\text{pass}} = f_{\text{pass}}/f_s = (f_s/2)/f_s = 0.5$, we obtain an equation for the 3-dB frequency F_0:

$$A_{\text{pass}} = 10\log_{10}\left[1 + \left(\frac{0.5}{F_0}\right)^{2N}\right] \quad \Rightarrow \quad F_0 = \frac{0.5}{\left(10^{A_{\text{pass}}/10} - 1\right)^{1/2N}}$$

The prefilter's stopband must begin at $F_{\text{stop}} = f_{\text{stop}}/f_s = (Lf_s - f_s/2)/f_s = L - 0.5$. Thus, the stopband attenuation will be

$$A_{\text{stop}} = 10\log_{10}\left[1 + \left(\frac{L-0.5}{F_0}\right)^{2N}\right] \quad \Rightarrow \quad L = 0.5 + F_0\left(10^{A_{\text{stop}}/10} - 1\right)^{1/2N}$$

For large A_{stop} we may ignore the -1 to get

$$L = 0.5 + F_0 10^{A_{\text{stop}}/20N}$$

With the values $A_{\text{pass}} = 0.1$ and $N = 3$, we find $F_0 = 0.94$. Fig. P12.18 shows a plot of L versus A_{stop}.

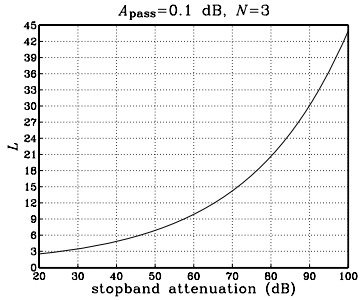


Fig. P12.18 Decimation ratio versus stopband attenuation in Problem 12.15.

Problem 12.16

Using the expression of the attenuation of Problem 12.15, we get at the passband and stopband normalized frequencies $F_{\text{pass}} = 0.5$, $F_{\text{stop}} = L - 0.5$:

$$\left.\begin{aligned} A_{\text{pass}} &= 10\log_{10}\left[1 + \left(\frac{0.5}{F_0}\right)^{2N}\right] \\ A_{\text{stop}} &= 10\log_{10}\left[1 + \left(\frac{L-0.5}{F_0}\right)^{2N}\right] \end{aligned}\right\} \quad \Rightarrow \quad \left(\frac{L-0.5}{0.5}\right)^{2N} = \frac{10^{A_{\text{stop}}/10} - 1}{10^{A_{\text{pass}}/10} - 1}$$

which may be solved for N, giving:

$$N = \frac{\ln\left(\dfrac{10^{A_{\text{stop}}/10} - 1}{10^{A_{\text{pass}}/10} - 1}\right)}{2\ln(2L-1)}$$

Let $N_0 = \text{ceil}(N)$. In order for the filter order to remain fixed at N_0, the computed quantity N must be in the range

$$N_0 - 1 < \frac{\ln\left(\dfrac{10^{A_{\text{stop}}/10} - 1}{10^{A_{\text{pass}}/10} - 1}\right)}{2\ln(2L-1)}$$

Solving this inequality for L, we find the limits on L:

$$0.5 + 0.5\left(\frac{10^{A_{\text{stop}}/10} - 1}{10^{A_{\text{pass}}/10} - 1}\right)^{1/2N_0} \le L < 0.5 + 0.5\left(\frac{10^{A_{\text{stop}}/10} - 1}{10^{A_{\text{pass}}/10} - 1}\right)^{1/2(N_0-1)} \le N_0$$

For the given numerical values, we find $N_0 = 3$ and the range $9.86 \le L < 40.97$, which gives the integer range $10 \le L \le 40$.

Problem 12.17

The normalized transition width is $\Delta F = \Delta f/f_s = (24.41 - 20)/44.1 = 0.1$. Using a Kaiser design, the transition width parameter D and filter length N will be:

$$D = \frac{A - 7.95}{14.36} = \frac{95 - 7.95}{14.36} = 6.0620, \qquad N - 1 = \frac{DL}{\Delta F} = \frac{6.062 \cdot 160}{0.1} = 9699.2 \quad \Rightarrow \quad N = 9920$$

where the final N is of the form $N = 2LK + 1 = 320K + 1$ with $K = 31$. Thus, the interpolation/decimation filter operating at rate $f_s'' = 160f_s = 147f_s' = 7.056$ MHz can be implemented with 160 polyphase subfilters, each of length $2K = 62$ and operating at a rate $f_s = 44.1$ kHz. The overall sample rate conversion cost in MAC/sec can be calculated from Eq. (12.6.9):

$$R = 2Kf_s' = 61f_s' = 61 \cdot 48 = 2928 \text{ kMAC/sec} = 2.928 \text{ MIPS}$$

The computational speed is easily handled by today's DSP chips. Memory, however, to store the N FIR coefficients is high—of the order of 10k words.

Problem 12.18

Here, we have the conversion ratio $f_s' = 32 = \frac{32}{48} \cdot 48 = \frac{2}{3} f_s$, so that $L = 2$, $M = 3$. Because $f_s' < f_s$, the filter's cutoff frequency will be $f_c = f_s'/2 = 16$ kHz. Assuming a ± 1 kHz transition width about f_c, that is, $\Delta f = 2$ kHz, and a $A = 60$ dB stopband attenuation, we find the Kaiser parameter D and filter length N:

$$D = \frac{A - 7.95}{14.36} = \frac{60 - 7.95}{14.36} = 3.6247, \qquad N-1 = \frac{DLf_s}{\Delta f} = \frac{3.6247 \cdot 2 \cdot 48}{2} = 173.98 \quad \Rightarrow \quad N = 177$$

where we rounded N to the next odd integer of the form $N = 2LK + 1$. Thus, $K = 44$, and there will be $L = 2$ polyphase subfilters of length $2K = 88$. Fig. P12.19 shows the ideal specifications of such a filter. The filter operates at the fast rate $f_s'' = 3f_s' = 2f_s = 96$ kHz and acts as an antialiasing filter for the downsampler, that is, it removes the high frequencies from the input in the range $[f_s'2, f_s/2] = [16, 24]$ kHz, so that the downshifted replicas—caused by the 3-fold downsampling operation—will not overlap.

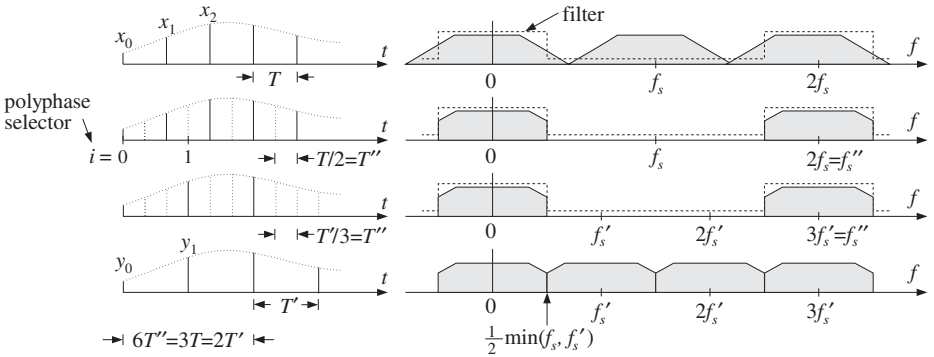


Fig. P12.19 Ideal SRC filter with conversion ratio 2/3.

Because $T = T''/2$ and $T' = T''/3$, it follows that the basic block interval will be $T_{\text{block}} = 6T'' = 3T = 2T'$, containing 3 input-rate samples $\{x_0, x_1, x_2\}$ and 2 output-rate samples $\{y_0, y_1\}$. The interpolated output y_1 that lies halfway between x_1 and x_2 is obtained by the polyphase filter $\mathbf{h}_1$. Indeed, the polyphase selector indices can be precomputed by

$$i_m = 3m\%2 = [0, 1], \qquad \text{for } m = 0, 1$$

The sample processing algorithm of the 2/3 sample rate converter will be then:

for each input block $\{x_0, x_1, x_2\}$ *do:*
- $w_0 = x_0$
- $y_0 = \text{dot}(P, \mathbf{h}_0, \mathbf{w}) = x_0$
- $\text{delay}(P, \mathbf{w})$
- $w_0 = x_1$
- $y_1 = \text{dot}(P, \mathbf{h}_1, \mathbf{w})$
- $\text{delay}(P, \mathbf{w})$
- $w_0 = x_2$
- $\text{delay}(P, \mathbf{w})$

where $P = 2K - 1$ is the order of the polyphase subfilters $\mathbf{h}_0$, $\mathbf{h}_1$, and $\mathbf{w}$ is the length-$(P + 1)$ input-rate delay-line buffer. Note that there is no interpolated output after x_2 (the next two outputs come from the next group of three inputs), and therefore, x_2 is simply read into the buffer $\mathbf{w}$ and the delay line is updated.

For reference, we also show in Fig. P12.20 the reverse converter by a ratio 3/2, converting from 32 kHz up to 48 kHz. Here, the SRC filter acts as an anti-image filter for the upsampler.

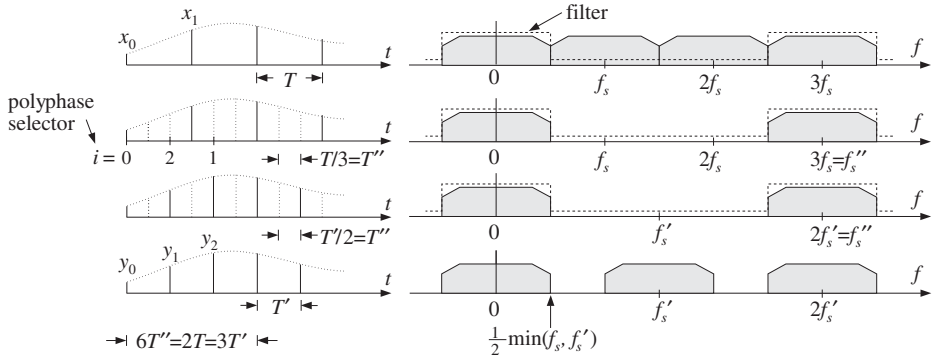


Fig. P12.20 Ideal SRC filter with conversion ratio 3/2.

In this case, we have $f_s' = 3f_s/2 = 3 \cdot 32/2 = 48$. The fast rate is $f_s'' = 2f_s' = 3f_s = 96$ kHz. The input- and output-rate sampling intervals are $T = 3T''$ and $T' = 2T''$, and the basic time block $T_{\text{block}} = 6T'' = 2T = 3T'$. Thus, every group of two input samples $\{x_0, x_1\}$ generates a group of three output samples $\{y_0, y_1, y_2\}$. As can be seen in the figure, the polyphase selector sequence is $i_m = 2m\%3 = [0, 2, 1]$, for $m = 0, 1, 2$. Therefore, the three interpolated outputs $\{y_0, y_1, y_2\}$ will be computed by the subfilters $\{\mathbf{h}_0, \mathbf{h}_2, \mathbf{h}_1\}$. The corresponding sample processing algorithm will be:

for each input block $\{x_0, x_1\}$ *do:*
- $w_0 = x_0$
- $y_0 = \text{dot}(P, \mathbf{h}_0, \mathbf{w}) = x_0$
- $y_1 = \text{dot}(P, \mathbf{h}_2, \mathbf{w})$
- $\text{delay}(P, \mathbf{w})$
- $w_0 = x_1$
- $y_2 = \text{dot}(P, \mathbf{h}_1, \mathbf{w})$
- $\text{delay}(P, \mathbf{w})$

Problem 12.19

For the case $L/M = 7/4$, we have $f_s' = 7f_s/4$ and $f_s'' = 4f_s' = 7f_s$, $T'' = T'/4 = T/7$, $T_{\text{block}} = 28T'' = 7T' = 4T$. Fig. P12.21 depicts the operation of the converter both in the time and frequency domains.

In the time domain, each input block of 4 input-rate samples $\{x_0, x_1, x_2, x_3\}$ generates 28 fast-rate interpolated samples, out of which 7 output-rate samples $\{y_0, y_1, y_2, y_3, y_4, y_4, y_6\}$ are selected and computed according to their polyphase indices:

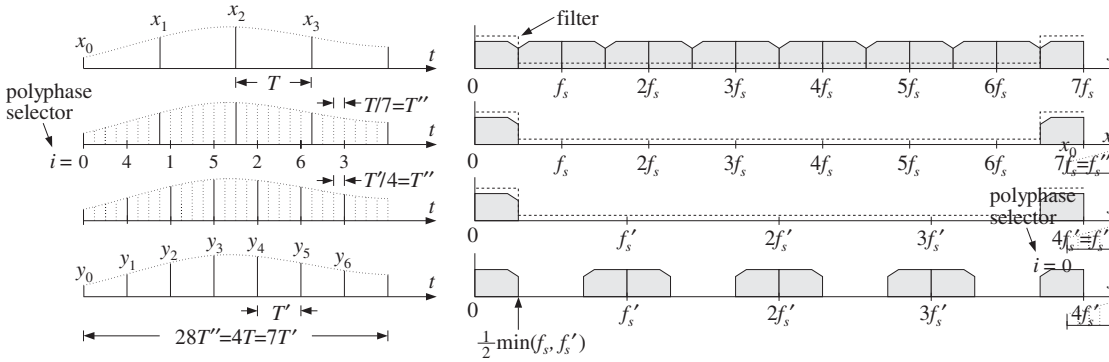


Fig. P12.21 Ideal SRC filter with conversion ratio 7/4.

$$i_m = 4m\%7 = [0,4,1,5,2,6,3], \qquad \text{for } m = 0,1,\ldots,6$$

$$n_m = (4m - i_m)/7 = [0,0,1,1,2,2,3]$$

whenever the index n_m is repeated, the input-rate polyphase delay line is not updated. The sample processing conversion algorithm is as follows:

for each input block $\{x_0, x_1, x_2, x_3\}$ *do:*
- $w_0 = x_0$
  - $y_0 = \text{dot}(P, \mathbf{h}_0, \mathbf{w}) = x_0$
  - $y_1 = \text{dot}(P, \mathbf{h}_4, \mathbf{w})$
- $\text{delay}(P, \mathbf{w})$
- $w_0 = x_1$
  - $y_2 = \text{dot}(P, \mathbf{h}_1, \mathbf{w})$
  - $y_3 = \text{dot}(P, \mathbf{h}_5, \mathbf{w})$
- $\text{delay}(P, \mathbf{w})$
- $w_0 = x_2$
  - $y_4 = \text{dot}(P, \mathbf{h}_2, \mathbf{w})$
  - $y_5 = \text{dot}(P, \mathbf{h}_6, \mathbf{w})$
- $\text{delay}(P, \mathbf{w})$
- $w_0 = x_3$
  - $y_6 = \text{dot}(P, \mathbf{h}_3, \mathbf{w})$
- $\text{delay}(P, \mathbf{w})$

For example, the two output samples y_4, y_5 lie between the input samples x_2, x_3 and correspond to the $i = 2$ and $i = 6$ interpolated samples. The delay line is updated after both outputs are computed, and then the next input sample x_3 is read into the delay line, and so on.

In the frequency domain, the SRC filter acts as an anti-image postfilter for the upsampler, removing the $L - 1 = 6$ replicas between multiples of the fast rate f_s''. The downshifting of the replicas caused by the downsampling operation will position replicas at the 3 multiples of the output rate f_s', $2f_s'$, and $3f_s'$.

In the $L/M = 4/7$ case, we have $f_s' = 4f_s/7$ and $f_s'' = 7f_s' = 4f_s$, $T'' = T'/7 = T/4$, $T_{\text{block}} = 28T'' = 4T' = 7T$. Fig. P12.22 depicts the operation of the converter both in the time and frequency domains.

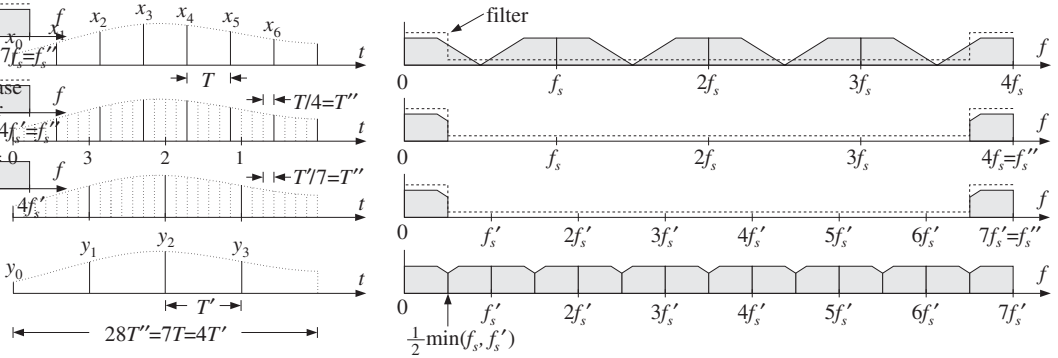


Fig. P12.22 Ideal SRC filter with conversion ratio 4/7.

In the time domain, each input block of 7 input-rate samples $\{x_0, x_1, x_2, x_3, x_4, x_5, x_6\}$ generates 28 fast-rate interpolated samples, out of which 4 output-rate samples $\{y_0, y_1, y_2, y_3\}$ are selected and computed according to their polyphase indices:

$$i_m = 7m\%4 = [0,3,2,1], \qquad \text{for } m = 0,1,\ldots,3$$

$$n_m = (7m - i_m)/4 = [0,1,3,5]$$

so that only the input samples x_0, x_1, x_3, x_5 will produce output samples. However, for the remaining input samples the delay line must be updated. Thus, the sample processing conversion algorithm will be as follows:

for each input block $\{x_0, x_1, x_2, x_3, x_4, x_5, x_6\}$ *do:*
- $w_0 = x_0$
  - $y_0 = \text{dot}(P, \mathbf{h}_0, \mathbf{w}) = x_0$
- $\text{delay}(P, \mathbf{w})$
- $w_0 = x_1$
  - $y_1 = \text{dot}(P, \mathbf{h}_3, \mathbf{w})$
- $\text{delay}(P, \mathbf{w})$
- $w_0 = x_2$
- $\text{delay}(P, \mathbf{w})$
- $w_0 = x_3$
  - $y_2 = \text{dot}(P, \mathbf{h}_2, \mathbf{w})$
- $\text{delay}(P, \mathbf{w})$ $\quad w_0 = x_4$
- $\text{delay}(P, \mathbf{w})$
- $w_0 = x_5$
  - $y_3 = \text{dot}(P, \mathbf{h}_1, \mathbf{w})$
- $\text{delay}(P, \mathbf{w})$
- $w_0 = x_6$
- $\text{delay}(P, \mathbf{w})$

In the frequency domain, the SRC filter acts as an antialiasing prefilter for the downsampler, removing the 3 replicas between multiples of the fast rate $f_s{}''$. The downshifting of the replicas caused by the downsampling operation will position replicas at the 6 multiples of the output rate $f_s{}', 2f_s{}', \ldots, 6f_s{}'$, without overlapping.

As a design example, consider the $4/7$ case and assume a Kaiser design with a stopband attenuation of 80 dB and normalized transition width $\Delta F = \Delta f / f_s = 0.1$. The width parameter D and filter length N will be:

$$D = \frac{A - 7.95}{14.36} = \frac{80 - 7.95}{14.36} = 5.0174\,, \qquad N-1 = \frac{DL}{\Delta F} = \frac{5.0174 \cdot 4}{0.1} = 200.70 \qquad \Rightarrow \qquad N = 209$$

where N the smallest odd integer of the form $N = 2LK + 1$; here, $K = 26$.

Problem 12.20

A typical window depends on the time variable k through the ratio $k/(N-1)$. The effective length of the stretched impulse response is $N_\rho - 1 = 2LK_\rho = 2LK/\rho$. A symmetric window of such length will depend on the variable $k''/(2LK_\rho) = \rho k''/(2LK)$. Thus, the window of length $2LK_\rho$ can be expressed in terms of the same window of length $2LK$, via the stretching relationship $w_\rho(k'') = w(\rho k'')$. It follows that the corresponding windowed impulse responses will be

$$w_\rho(k'')\, d_\rho(k'') = \rho\, w(\rho k'')\, d(\rho k'')$$

Problem 12.21

The polyphase Kaiser filter can be designed by the code fragment:

```
double *hd, **h, *w, *x, *y, wind, pi = 4 * atan(1.0);

D = (A - 7.95) / 14.36;

if (A > 50)
       alpha = 0.1102 * (A - 8.7);
else if (A > 21)
       alpha = 0.5842 * pow(A-21, 0.4) + 0.07886 * (A-21);
else
       alpha = 0;

N = 1 + ceil(D*L / DF);                                  Kaiser length
K = ceil((N - 1) / (2.0*L));                             round up
N = 2*L*K + 1;                                           final length
P = 2*K - 1;                                             polyphase filter order
LK = L*K;                                                filter delay

hd = (double *) calloc(N, sizeof(double));               allocate direct form

wc = pi / max(L, M);                                     cutoff frequency
I0alpha = I0(alpha);                                     window normalization

for (n=0; n<N; n++)                                      compute filter
       if (n==LK)                                        middle value
              hd[n] = L /(double)  max(L, M);
       else   {
              wind = I0(alpha * sqrt(n*(2.0*LK-n)) / LK) / I0alpha;
              hd[n] = wind * L * (sin(wc*(n-LK))) / (pi*(n-LK));
              }

h = (double **) calloc(L, sizeof(double *));             allocate polyphase
for (i = 0; i < L; i++)
       h[i] = (double *) calloc(P+1, sizeof(double));    i-th polyphase

for (i=0; i<L; i++)                                      define polyphase filters
       for (n=0; n<=P; n++)
              h[i][n] = hd[L*n+i];
```

Assuming the input data are being read from `stdin`, we allocate and initialize the polyphase delay-line buffer **w** as follows:

```
w = (double *) calloc(P+1, sizeof(double));              allocate w

for (i=K; i>=1; i--)                                     read K input samples
       if (scanf("%lf", w+i) == EOF) {
            printf("must have at least K = %d input samples\n", K);
            exit(0);
            }
```

Then, allocate the input/output blocks for the algorithm Eq. (12.6.13) and keep processing input samples in groups of M. The computed outputs go into `stdout`. Upon encountering the end-of-file of the inputs, we jump out of the for-ever loop and call Eq. (12.6.13) K/M more times with zero input samples. Because each call generates M zero inputs, the K/M calls will generate approximately $M(K/M) = K$ zero inputs:

```
x = (double *) calloc(M, sizeof(double));
y = (double *) calloc(L, sizeof(double));

for (;;) {                                               for-ever do:
       for (n=0; n<M; n++)                               read M inputs
              if (scanf("%lf", x+n) == EOF)
                     goto transients;
       src(L, M, P, h, w, x, y);                         compute L outputs
       for (m=0; m<L; m++)                               write L outputs
              printf("%.12lf\n", y[m]);
       }

transients:

for (i=0; i <= K/M; i++) {                               last K inputs
       for (n=0; n<M; n++)
              x[n] = 0;
       src(L, M, P, h, w, x, y);
       for (m=0; m<L; m++)
              printf("%.12lf\n", y[m]);
       }
```

The sample processing algorithm (12.6.13) is implemented by the routine:

```
/* src.c - sample rate converter processing algorithm */

#include <math.h>

double dot();
```

```
void delay();

void src(L, M, P, h, w, x, y)              h is Lx(P+1) polyphase filter matrix
double **h, *w, *x, *y;                    w is (P+1)-dimensional delay buffer
{                                          x,y are M- and L-dimensional vectors
       int n, m, i;
       double R = L / (double) M;                          conversion ratio

       for (n = 0; n < M; n++) {
              w[0] = x[n];
              for (m = ceil(R*n); m <= floor(R*(n+1)); m++) {
                     i = (M * m) % L;                      polyphase selector
                     y[m] = dot(P, h[i], w);
                     }
              delay(P, w);
              }
}
```

With the values $A = 30$ dB, $\Delta F = 0.1$, the 5/3 filter has $D = 1.536$, $\alpha = 2.117$, $N = 81$, $K = 8$, $P = 2K - 1 = 15$. The reverse 3/5 filter has the same D, α, K, and P, but its length is $N = 49$, because $N = 2LK + 1$, where now $L = 3$. The filter responses are shown in Fig. P12.23 both in absolute and dB scales. The dc gains are 5 and 3, respectively. The dB responses have been normalized to 0 dB at DC.

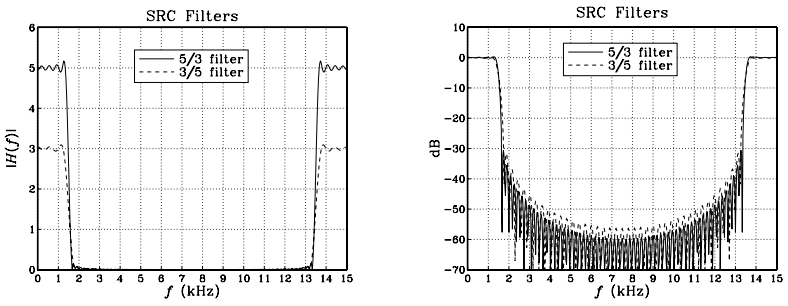


Fig. P12.23 Frequency responses of SRC filters.

Both filters have a cutoff frequency of $f_c = \min(f_s/2, f_s'/2) = 3/2 = 1.5$ kHz, or, $\omega_c'' = \pi/5$. The passband ripple is $\delta = 10^{-A/20} = 0.032$. Their transition widths are different: the first filter has $\Delta f = 0.1\times 3 = 0.3$ kHz, and the second $\Delta f = 0.1\times 5 = 0.5$ kHz. Were they to be redesigned to have the same Δf, say 0.3 kHz, then for the 3/5 filter we would have $\Delta F = 0.06$, resulting in the lengths $N = 79$, $K = 13$, $P = 25$. In that case, the filters would be virtually identical (except for the gain and the delay by 1 sample).

The impulse responses are obtained from Eq. (12.6.5), with $LK = 5\times 8 = 40$ and $LK = 3\times 8 = 24$ in the two cases, that is,

$$h_{5/3}(n'') = 5\,\frac{I_0(\alpha\sqrt{n''(80-n'')}/80)}{I_0(\alpha)}\,\frac{\sin(\pi(n''-40)/5)}{\pi(n''-40)}, \quad n'' = 0,1,\ldots,80$$

$$h_{3/5}(n'') = 3\,\frac{I_0(\alpha\sqrt{n''(48-n'')}/48)}{I_0(\alpha)}\,\frac{\sin(\pi(n''-24)/5)}{\pi(n''-24)}, \quad n'' = 0,1,\ldots,48$$

The frequency responses were computed by direct evaluation of:

$$H(f) = \sum_{n''=0}^{N-1} h(n'')\,e^{-2\pi j f n''/f_s''}, \qquad \text{with } f_s'' = 15 \text{ kHz}$$

Figure P12.24 shows the output $y(n')$ of the 5/3 filter and compares it with the signal $x'(n')$ that would have been obtained had the analog signal been sampled at 3 kHz. On the right, it shows the effect of the reverse 3/5 filter on $y(n')$, and compares it with the original $x(n)$.

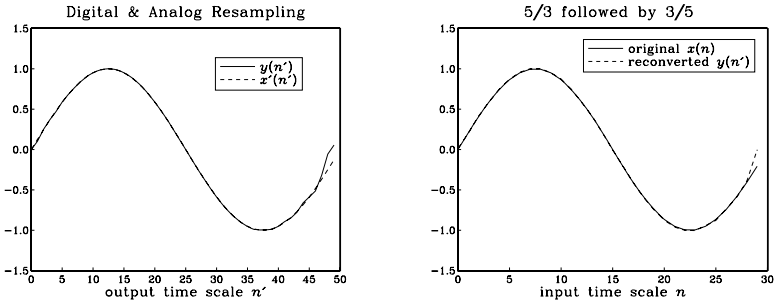


Fig. P12.24 Comparison of $y(n')$ and $x'(n')$, and of $x(n)$ and reconverted $y(n')$.

Problem 12.22

Replacing the quantizer Q by its equivalent noise model, we can write the ζ-domain input/output equations:

$$V'(\zeta) = X'(\zeta) - \zeta^{-1}Y'(\zeta)$$

$$W_0'(\zeta) = V'(\zeta) + \zeta^{-1}W_0'(\zeta)$$

$$Y'(\zeta) = E'(\zeta) + W_0'(\zeta)$$

Eliminating $V'(\zeta)$ and $W_0'(\zeta)$, we obtain:

$$Y'(\zeta) = X'(\zeta) + (1-\zeta^{-1})E'(\zeta)$$

Therefore, $H_x(\zeta) = 1$, and $H_{NS}(\zeta) = 1 - \zeta^{-1}$. The main advantage of this form is that the input signal $X'(\zeta)$ appears unchanged at the output (as opposed to being delayed as in the conventional case).

Problem 12.23

Using the signal labels indicated on the block diagram, we have the sample processing algorithm:

for each input x do:
$v = x - u$
$w_0 = v + w_1$
$y = Q(w_0)$
$w_1 = w_0$
$u = y$

Iterating this algorithm 10 times, we get for $x = 0.4$ and $x = -0.2$:

x	y	x	y
0.40	1	−0.20	−1
0.40	−1	−0.20	1
0.40	1	−0.20	−1
0.40	1	−0.20	1
0.40	1	−0.20	−1
0.40	−1	−0.20	−1
0.40	1	−0.20	1
0.40	1	−0.20	−1
0.40	−1	−0.20	1
0.40	1	−0.20	−1

The averages of the y values are 0.4 and −0.2, respectively.

Problem 12.24

Working in the ζ-domain, we note that the input and output of the filter $H_1(\zeta)$ will be:

$$X'(\zeta) - \zeta^{-1}Y'(\zeta) \qquad \text{and} \qquad H_1(\zeta)\left(X'(\zeta) - \zeta^{-1}Y'(\zeta)\right)$$

Similarly, the input and output of $H_2(\zeta)$ are:

$$H_1(\zeta)\left(X'(\zeta) - \zeta^{-1}Y'(\zeta)\right) - \zeta^{-1}Y'(\zeta) \qquad \text{and} \qquad H_2(\zeta)\left[H_1(\zeta)\left(X'(\zeta) - \zeta^{-1}Y'(\zeta)\right) - \zeta^{-1}Y'(\zeta)\right]$$

Adding $E'(\zeta)$ to this output will generate $Y'(\zeta)$. Thus,

$$Y'(\zeta = E'(\zeta) + H_2(\zeta)\left[H_1(\zeta)\left(X'(\zeta) - \zeta^{-1}Y'(\zeta)\right) - \zeta^{-1}Y'(\zeta)\right]$$

Moving the $Y'(\zeta)$ to the left-hand side and solving for $Y'(\zeta)$ gives the transfer relationship:

$$Y'(\zeta) = \frac{H_1(\zeta)H_2(\zeta)}{1 + \zeta^{-1}H_2(\zeta) + \zeta^{-1}H_1(\zeta)H_2(\zeta)}\,X'(\zeta) + \frac{1}{1 + \zeta^{-1}H_2(\zeta) + \zeta^{-1}H_1(\zeta)H_2(\zeta)}\,E'(\zeta)$$

Thus, we identify:

$$H_x(\zeta) = \frac{H_1(\zeta)H_2(\zeta)}{1 + \zeta^{-1}H_2(\zeta) + \zeta^{-1}H_1(\zeta)H_2(\zeta)}, \qquad H_{NS}(\zeta) = \frac{1}{1 + \zeta^{-1}H_2(\zeta) + \zeta^{-1}H_1(\zeta)H_2(\zeta)}$$

The requirement that $H_x(\zeta) = 1$ and $H_{NS}(\zeta) = (1 - \zeta^{-1})^2$ is satisfied by the choices:

$$H_1(\zeta) = H_2(\zeta) = \frac{1}{1 - \zeta^{-1}}$$

A block diagram realization is shown in Fig. P12.25. The corresponding sample processing algorithm can be stated as follows:

for each input x′ do:
$w_0 = w_1 + (x' - u)$
$v_0 = v_1 + (w_0 - u)$
$y' = Q(v_0)$
$w_1 = w_0$
$v_1 = v_0$
$u = y'$

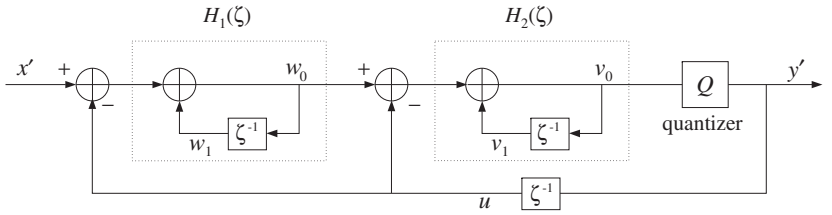


Fig. P12.25 Discrete-time model of second-order delta-sigma quantizer.

Problem 12.25

Without the feedback delay, we get a similar transfer relationship as in the previous problem, but without the ζ^{-1} delays in the denominators, that is,

$$Y'(\zeta) = \frac{H_1(\zeta)H_2(\zeta)}{1 + H_2(\zeta) + H_1(\zeta)H_2(\zeta)}\,X'(\zeta) + \frac{1}{1 + H_2(\zeta) + H_1(\zeta)H_2(\zeta)}\,E'(\zeta)$$

Thus, we identify:

$$H_x(\zeta) = \frac{H_1(\zeta)H_2(\zeta)}{1 + H_2(\zeta) + {}_1(\zeta)H_2(\zeta)}, \qquad H_{NS}(\zeta) = \frac{1}{1 + H_2(\zeta) + H_1(\zeta)H_2(\zeta)}$$

The requirement that $H_x(\zeta) = \zeta^{-1}$ and $H_{NS}(\zeta) = (1 - \zeta^{-1})^2$ is satisfied by the choices:

$$H_1(\zeta) = \frac{1}{1 - \zeta^{-1}}, \qquad H_2(\zeta) = \frac{\zeta^{-1}}{1 - \zeta^{-1}}$$

(This problem is similar to Problem 7.2.)

Problem 12.26

Once the decimation filters are designed, the 2-level quantization of the input by the delta-sigma quantizer, and the subsequent filtering by the decimation filter can be carried out by the following for-loop (which does not discard the interpolated outputs, as discussed in Example 12.7.2):

```
w1 = 0;                                              initialize delay

for (n=0; n<Ntot; n++) {
     x = 0.5 * sin(2 * pi * f0 * n);                 high-rate input sample
```

```
y = Q(w1);                          ΔΣ quantizer output
v = x - y;                          delta part
w0 = w1 + v;                        sigma part
w1 = w0;                            update quantizer's delay
yup[n] = fir(N-1, h, w, y);         (upsampled) decimator output
}
```

where **h** and **w** are the N-dimensional impulse response and internal state arrays for the decimator. In the Kaiser and the rectangular window designs, the filter length is $N = 2LM + 1$. In the first-order comb case, it is $N = 10$, and the impulse response and transfer function are:

$$\mathbf{h} = \frac{1}{10}[1,1,1,1,1,1,1,1,1,1]\,, \qquad H(\zeta) = \frac{1}{10}\,\frac{1-\zeta^{-10}}{1-\zeta^{-1}}$$

For the second-order comb case, we obtain the impulse response by convolving the first-order comb's impulse response with itself. This gives a length $N = 19$ response and transfer function:

$$\mathbf{h} = \frac{1}{100}[1,2,3,4,5,6,7,8,9,10,9,8,7,6,5,4,3,2,1]\,, \qquad H(\zeta) = \left[\frac{1}{10}\,\frac{1-\zeta^{-10}}{1-\zeta^{-1}}\right]^2$$

The magnitude response of the second-order comb case is the square of that of the first-order case:

$$|H(f)| = \left|\frac{\sin(\pi f/f_s)}{L\sin(\pi f/10f_s)}\right|^2$$

It is plotted in dB in Fig. P12.26 together with the first-order comb case and the rectangular window design. The upsampled output $y_{\text{up}}(n')$ of the second-order comb decimator is plotted in the right graph of Fig. P12.26.

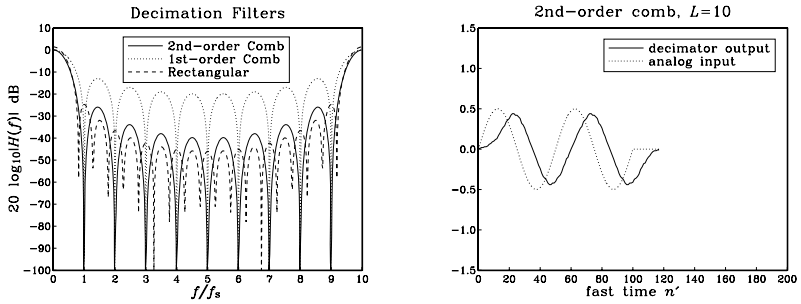


Fig. P12.26 Length-19 second-order comb magnitude response and filter output.

Problem 12.27

The filter designs are the same as in the previous problems. The implementation of the second-order delta-sigma quantizer is given by the sample processing algorithm discussed in Problem 12.24. The following for-loop generates the high-rate input, quantizes it by the 2nd-order quantizer, and filters the quantized output with the decimation filter:


```
w1 = v1 = u = 0;                    initialize delays

for (n=0; n<Ntot; n++) {
     x = 0.5 * sin(2 * pi * f0 * n);   high-rate input sample
     w0 = w1 + x - u;               output of H1
     v0 = v1 + w0 - u;              output of H2
     y = Q(v0);                     ΔΣ quantizer output
     w1 = w0;                       update quantizer's delays
     v1 = v0;                       update quantizer's delays
     u = y;                         update quantizer's delays
     yup[n] = fir(N-1, h, w, y);    (upsampled) decimator output
     }
```

Figure P12.27 shows the signal $x'(n')$ and the 2-level quantized output $y'(n')$ of the second-order quantizer.

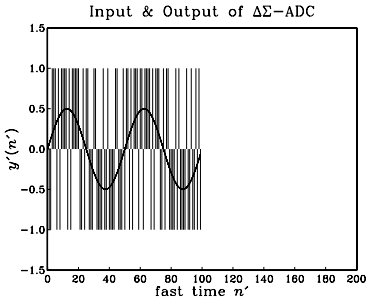


Fig. P12.27 High-rate input and output signals of second-order ΔΣ quantizer.

The upsampled outputs $y_{\text{up}}(n')$ of the averaging, second-order comb, rectangular, and Kaiser window decimators are shown in Fig. P12.28.

Problem 12.28

Working in the ζ-domain, we obtain the I/O equation of the first stage. The input to $H(\zeta)$ is $X'(\zeta) - \zeta^{-1}Y_1'(\zeta)$, Thus, adding the noise component, we have:

$$Y_1'(\zeta) = E_1'(\zeta) + H(\zeta)\left[X'(\zeta) - \zeta^{-1}Y_1'(\zeta)\right]$$

which can be rearranged as:

$$Y_1'(\zeta) = \frac{H(\zeta)}{1+\zeta^{-1}H(\zeta)}\,X'(\zeta) + \frac{1}{1+\zeta^{-1}H(\zeta)}\,E_1'(\zeta)$$

Inserting $H(\zeta) = 1/(1-\zeta^{-1})$, we obtain the transfer relationship of one stage:

$$Y_1'(\zeta) = X'(\zeta) + (1-\zeta^{-1})E_1'(\zeta) = X'(\zeta) + D(\zeta)E_1'(\zeta) \qquad \text{(P12.3)}$$

where $D(\zeta) = 1-\zeta^{-1}$. The input to the second stage is obtained by forming the difference of the signals around the first quantizer, that is, $(Y' - E_1') - Y' = -E_1'$. Applying the basic I/O equation (P12.3) to this input and corresponding noise E_2', we obtain

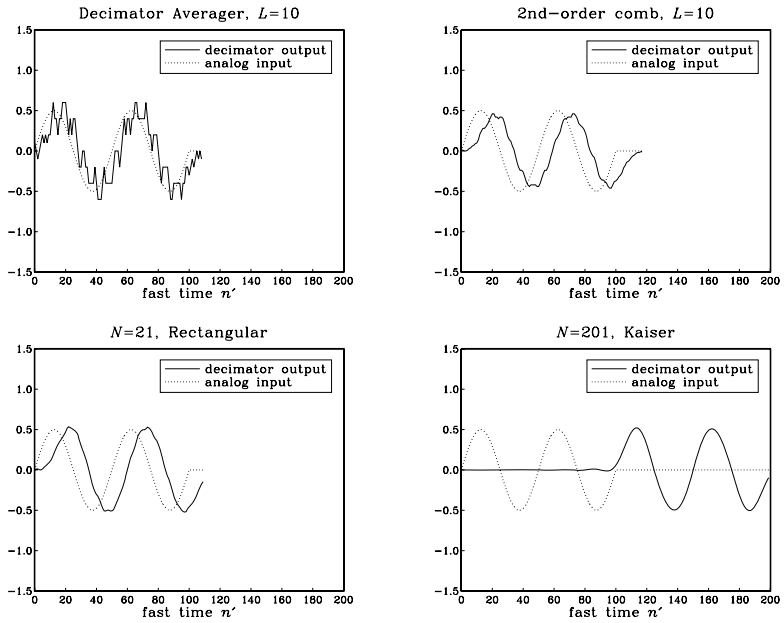


Fig. P12.28 First- and second-order comb, and rectangular- and Kaiser-window decimator outputs.

$$Y_2'(\zeta) = -E_1'(\zeta) + D(\zeta)E_2'(\zeta) \tag{P12.4}$$

The overall output is obtained by the linear combination:

$$\begin{aligned} Y'(\zeta) &= Y_1'(\zeta) + D(\zeta)Y_2'(\zeta) = X'(\zeta) + D(\zeta)E_1'(\zeta) + D(\zeta)\left[-E_1'(\zeta) + D(\zeta)E_2'(\zeta)\right] \\ &= X'(\zeta) + D(\zeta)E_1'(\zeta) - D(\zeta)E_1'(\zeta) + D^2(\zeta)E_2'(\zeta) = X'(\zeta) + D^2(\zeta)E_2'(\zeta) \end{aligned}$$

Thus, the quantization error from the first stage is canceled, whereas the error of the second stage gets filtered by the second-order highpass filter $D^2(\zeta) = (1-\zeta^{-1})^2$.

Problem 12.29

Such an arrangement is shown in Fig. P12.29. The outputs of the three stages are:

$$\begin{aligned} Y_1'(\zeta) &= X'(\zeta) + D(\zeta)E_1'(\zeta) \\ Y_2'(\zeta) &= -E_1'(\zeta) + D(\zeta)E_2'(\zeta) \\ Y_3'(\zeta) &= -E_2'(\zeta) + D(\zeta)E_3'(\zeta) \end{aligned}$$

The overall output is obtained by the linear combination:

$$\begin{aligned} Y'(\zeta) &= Y_1'(\zeta) + D(\zeta)\left[Y_2'(\zeta) + D(\zeta)Y_3'(\zeta)\right] \\ &= X'(\zeta) + D(\zeta)E_1'(\zeta) + D(\zeta)\left[-E_1'(\zeta) + D(\zeta)E_2'(\zeta) + D(\zeta)\left(-E_2'(\zeta) + D(\zeta)E_3'(\zeta)\right)\right] \\ &= X'(\zeta) + D^3(\zeta)E_3'(\zeta) \end{aligned}$$

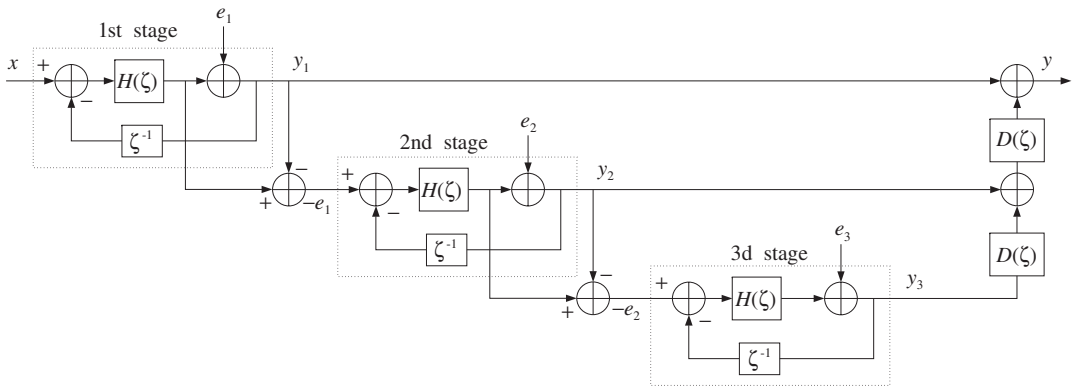


Fig. P12.29 MASH architecture of third-order delta-sigma quantizer.

Problem 12.30

Such an architecture is depicted in Fig. P12.30. The I/O relationship of the first stage is

$$Y_1'(\zeta) = X'(\zeta) + D(\zeta)E_1'(\zeta)$$

The input to the second-order quantizer is $-E_1'(\zeta)$. Therefore, its I/O relationship is (from Problem 12.8.3):

$$Y_2'(\zeta) = -E_1'(\zeta) + D^2(\zeta)E_2'(\zeta)$$

The overall output is obtained by the linear combination:

$$\begin{aligned} Y'(\zeta) &= Y_1'(\zeta) + D(\zeta)Y_2'(\zeta) = X'(\zeta) + D(\zeta)E_1'(\zeta) + D(\zeta)\left[-E_1'(\zeta) + D^2(\zeta)E_2'(\zeta)\right] \\ &= X'(\zeta) + D^3(\zeta)E_2'(\zeta) \end{aligned}$$

Problem 12.31

The delay of $(N-1)/2 = LM$ introduced by the decimation filter must be taken into account in both multiplexing and control applications. In feedback control applications, such a delay may render the closed-loop system unstable. See [294] for further discussion.

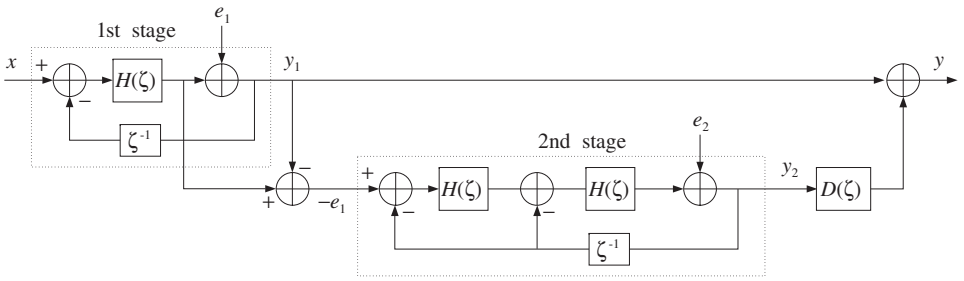


Fig. P12.30 MASH architecture of third-order delta-sigma quantizer.

Appendix B Problems

Problem B.1

The input sequence is

$$x(n) = \sum_i u_i \delta(n - iD - d)$$

Its autocorrelation function is

$$R_{xx}(k) = E[x(n+k)x(n)] = \sum_i \sum_j \sum_{d=0}^{D-1} E[u_i u_j]\delta(n+k-iD-d)\delta(n-jD-d)p(d)$$

where the expectation values were taken with respect to both u_i and d. Because the u_i are mutually independent and zero mean and d is uniform, we have $E[u_i u_j] = \sigma_u^2 \delta(i-j)$ and $p(d) = 1/D$. It follows that:

$$R_{xx}(k) = \frac{\sigma_u^2}{D} \sum_i \sum_{d=0}^{D-1} \delta(n+k-iD-d)\delta(n-iD-d)$$

Changing variables to $m = n - iD - d$ and noting that m runs over all the integers, we can replace the double summation over i and d by the summation over m:

$$R_{xx}(k) = \frac{\sigma_u^2}{D} \sum_m \delta(k+m)\delta(m) = \frac{\sigma_u^2}{D}\delta(k)$$

Thus, $x(n)$ is stationary and has a delta-function autocorrelation. Sending this sequence into the interpolation filter will generate a stationary output sequence $y(n)$ with a noise reduction ratio:

$$NRR = \frac{\sigma_y^2}{\sigma_x^2} = \sum_n h(n)^2$$

For the particular case of a hold interpolator, $h(n)$ consists of D ones:

$$NRR = \frac{\sigma_y^2}{\sigma_x^2} = \sum_{n=0}^{D-1} 1 = D \qquad \Rightarrow \qquad \sigma_y^2 = D\sigma_x^2 = D\frac{\sigma_u^2}{D} = \sigma_u^2$$

For the linear interpolator, we have $h(n) = 1 - |n|/D$, for $|n| \le D$. This gives:

$$NRR = \frac{\sigma_y^2}{\sigma_x^2} = \sum_{n=0}^{D} \left(1 - \frac{|n|}{D}\right)^2 = 1 + 2\sum_{n=1}^{D} \left(1 - \frac{n}{D}\right)^2 = 1 + 2\frac{1}{D^2}\sum_{m=1}^{D-1} m^2$$

Using the identity $\sum_{m=1}^{D-1} m^2 = \frac{1}{6}D(D-1)(2D-1)$, we obtain:

$$NRR = \frac{\sigma_y^2}{\sigma_x^2} = 1 + 2\sum_{m=1}^{D-1} m^2 = 1 + \frac{D(D-1)(2D-1)}{3D^2} = 1 + \frac{(D-1)(2D-1)}{3D} = \frac{2D^2+1}{3D}$$

Thus,

$$\sigma_y^2 = \frac{2D^2+1}{3D}\sigma_x^2 = \frac{2D^2+1}{3D^2}\sigma_u^2$$

Problem B.2

The power spectral density of the output sequence will be related to that of the input by:

$$S_{yy}(z) = H(z)H(z^{-1})S_{xx}(z) = H(z)H(z^{-1})\sigma_x^2 = H(z)H(z^{-1})\frac{\sigma_u^2}{D}$$

For the hold interpolator, we have:

$$H(z) = \frac{1-z^{-D}}{1-z^{-1}}, \qquad H(z^{-1}) = \frac{1-z^{D}}{1-z} = \frac{1-z^{-D}}{1-z^{-1}}\,z^{D-1}$$

Thus, the output psd becomes:

$$S_{yy}(z) = H(z)H(z^{-1})\frac{\sigma_u^2}{D} = \frac{1}{D}\left[\frac{1-z^{-D}}{1-z^{-1}}\right]^2 z^{D-1}\sigma_u^2$$

Taking inverse z-transforms with the help of Problem 5.11, we find:

$$R_{yy}(k) = \left(1 - \frac{|k|}{D}\right)\sigma_u^2$$

Problem B.3

The generation of the held signal and the computation of its autocorrelation is illustrated by the following C program segment:

```
double *u, *y, *R;
long iseed;

y = (double *) calloc(N, sizeof(double));          N-dimensional
R = (double *) calloc(M+1, sizeof(double));        (M+1)-dimensional

u[0] = ran(&iseed)-0.5;                            initialize u
q = D * ran(&iseed);                               initialize q

for (n=0; n<N; n++)
       y[n] = ranh(D, u, &q, &iseed);              zero mean

corr(N, y, y, M, R);                               sample autocorrelation
```

A typical sample autocorrelation is shown in Fig. P14.1

Problem B.4

As was mentioned in the end of Section B.3, the averaged periodogram spectrum can be calculated by the following loop:

```
for (k=0; k<K; k++) {                              generate K = 200 blocks
    u[0] = ran(&iseed);                            initialize kth block
    q = D * ran(&iseed);

    for (n=0; n<N; n++)                            generate kth block
        X[n] = cmplx(ranh(D, u, &q, &iseed), 0.0);

    fft(N, X);                                     compute its FFT

    for(n=0; n<N; n++)                             accumulate its periodogram
        S[i] += cabs(X[i]) * cabs(X[i]);
    }
```

A typical averaged spectrum is shown in Fig. P14.2, for the case $D = 5$.

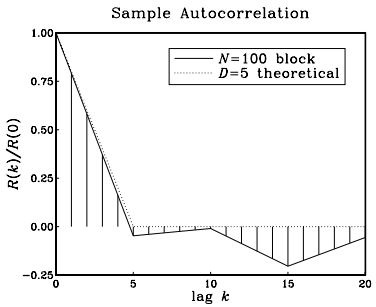


Fig. P14.1 Sample autocorrelation of length-100 hold noise.

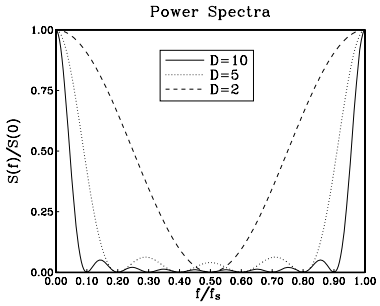


Fig. P14.2 Averaged periodogram spectrum of held noise.

Problem B.5

The random number generator C routine is:

```
/* rani.c - low-frequency random number generator using interpolation */

double ran(), dot();
void delay(), cdelay2();

double rani(L, P, h, w, q, iseed)          usage: y=rani(L, P, h, w, & q, & iseed);
int L, P, *q;                              q and iseed passed by reference
double **h, *w;                            h = polyphase filter matrix
long *iseed;                               required long by ran()
{
       int i;
       double y;

       if (*q==0)                          every L calls get new random number
              w[0] = ran(iseed) - 0.5;     zero mean

       i = (L - *q) % L;                   current polyphase index
       y = dot(P, h[i], w);                compute interpolated value
```

```
        cdelay2(L-1, q);                    decrement q for next call

        if (*q==0)                          every L calls update delay line
                delay(P, w);

        return y;
}
```

For the hold and linear interpolators, it generates the same output as `ranh.c` and `ranl.c`. The following program `raniex.c` is an example C program that invokes `rani`. Most of it is concerned with designing and allocating the polyphase filters. The very last for-loop generates the desired numbers. The for-loop before the last one, initializes the delay line $\mathbf{w}$ in accordance with Eq. (12.2.19):

```
/* raniex.c - interpolated random number generator example */

run with:
k = 0,1,2 = hold, linear, Kaiser interpolators
L = 10, Ntot = 150, iseed = 1234567
A = 30 dB, ΔF = 0.3


#include <stdio.h>
#include <math.h>
#include <stdlib.h>

double ran(), rani(), I0();                          I0.c was given in Ch.10

void main(int argc, char ** argv)
{
        double pi = 4 * atan(1.0);
        double **h, *w, *hd;
        double I0al, wind, A, D, wc, DF, al;
        int L, P, K, N, LK;
        int q, i, k, n, Ntot;
        long iseed;

        if (argc != 5) {
                puts("\nUsage: raniex k L Ntot iseed > y.dat\n"
                     "   k     = 0,1,2 for hold,linear,general interpolator\n"
                     "   L     = upsampling ratio\n"
                     "   Ntot  = total random numbers\n"
                     "   iseed = initial seed\n");
                exit(0);
                }

        k     = atoi(argv[1]);
        L     = atoi(argv[2]);
        Ntot  = atoi(argv[3]);
        iseed = atol(argv[4]);

        if (k >= 2) {                                        Kaiser interpolator
                fprintf(stderr, "enter A, DF (in units of fs): "  );
                scanf("%lf %lf", &A, &DF);

                D = (A - 7.95) / 14.36;
```



```
                if (A > 50)
                        al = 0.1102 * (A - 8.7);
                else if (A > 21)
                        al = 0.5842 * pow(A-21, 0.4) + 0.07886 * (A-21);
                else
                        al = 0;

                N = 1 + ceil(D * L / DF);

                K = ceil((N - 1) / (2.0*L));
                N = 2 * L * K + 1;                           filter length
                P = 2 * K - 1;                               filter order
                LK = L * K;                                  filter delay

                hd = (double *) calloc(N, sizeof(double));   direct form filter

                wc = pi / L;                                 ideal cutoff frequency

                I0al = I0(al);

                for (n=0; n<N; n++)                          windowed filter
                        if (n==LK)
                                hd[n] = 1;
                        else   {
                                wind = I0(al * sqrt(n*(2.0*LK-n)) / LK) / I0al;
                                hd[n] = wind * L * (sin(wc*(n-LK)))/(pi*(n-LK));
                                }
                }

        if (k == 0) {                                        hold interpolator
                N = L;
                K = 0;
                P = 0;
                for (n=0; n<N; n++)
                        hd[n] = 1;
                }

        if (k == 1) {                                        linear interpolator
                P = 1;
                K = 1;
                N = 2*L+1;
                for (n=0; n<N; n++)
                        hd[n] = 1 - abs(n - L) / (double) L;
                }


        h = (double **) calloc(L, sizeof(double *));         polyphase subfilters
        for (i = 0; i < L; i++)                              allocate L subfilters
                h[i] = (double *) calloc(P+1, sizeof(double));

        for (i=0; i<L; i++)
                for (n=0; n<=P; n++)
                        h[i][n] = hd[L*n+i];                 ith subfilter

        w = (double *) calloc(P+1, sizeof(double));          low-rate delay-line buffer

        for (i=K; i>=1; i--)
                w[i] = ran(&iseed) - 0.5;                    initialize w
```

```
        q = 0;

        for (n=0; n<Ntot; n++)                                  generate interpolated noise
              printf("%.12lf\n", rani(L, P, h, w, &q, &iseed));
}
```

With the choices $A = 30$ dB, $\Delta F = 0.3$, the Kaiser parameters become $D = 1.536$, $\alpha = 2.117$. The generated sequences are shown in Fig. P14.3.

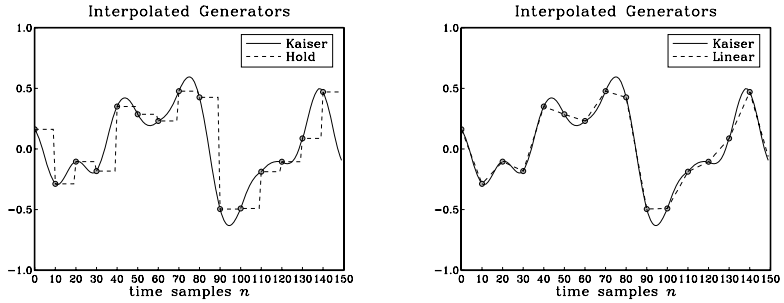


Fig. P14.3 Kaiser, hold, and linearly interpolated random sequences.

Problem B.6

Noting that the autocorrelation of a hold-interpolated sequence of period D is $R(k) = (1 - |k|/D)\sigma_u^2$, $-D \le k \le D$, we have:

$$\boxed{R_{yy}(k) = \frac{1}{B^2}\sum_{b=0}^{B-1} R_{y_b y_b}(k) = \frac{1}{B^2}\sum_{b=0}^{B-1}\left(1 - \frac{|k|}{2^b}\right) u(2^b - |k|)\,\sigma_u^2} \qquad \text{(P14.1)}$$

where the unit-step $u(2^b - |k|)$ restricts the duration of the b-th term to be $|k| \le 2^b$, or $|k| \le 2^b - 1$ for non-zero values of $R_{yy}(k)$. The resulting autocorrelation function is piece-wise linear, where the linear segments have geometrically increasing slopes. This is shown in Fig. P14.4 for the case $B = 4$. At lag $k = 0$, Eq. (P14.1) gives the expected result:

$$\sigma_y^2 = R_{yy}(0) = \frac{1}{B^2} \cdot B\sigma_u^2 = \frac{\sigma_u^2}{B}$$

The increasing linear slopes in $R_{yy}(k)$ cause it to have long-range correlations. Indeed, because of the restriction $|k| \le 2^b - 1$, the maximum lag will be:

$$k_{\max} = 2^{B-1} - 1 \qquad \text{(P14.2)}$$

Given a lag k in the range $|k| \le k_{\max}$, the expression of Eq. (P14.1) can be summed in closed form as follows. The restriction $|k| \le 2^b - 1$ can be turned around into a restriction for the summation index $b \ge \log_2(|k| + 1)$. Denoting the ceiling of this quantity by

$$b(k) = \lceil \log_2(|k| + 1) \rceil$$

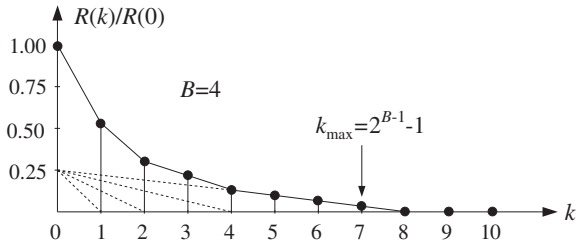


Fig. P14.4 Theoretical autocorrelation of $1/f$-noise model.

we find

$$R_{yy}(k) = \frac{1}{B^2}\sum_{b=b(k)}^{B-1}\left(1 - |k|2^{-b}\right)\sigma_u^2$$

and using the finite geometric series formula (e.g., Eq. (6.3.13)), we get the expression of Eq. (B.25).

Problem B.7

Figure P14.5 shows the theoretical autocorrelation (P14.1) or (B.25), together with the sample autocorrelation of a (zero-mean) block of $1/f$-noise samples of length $N = 2000$ for $B = 8$. The maximum correlation length is in this case $k_{\max} = 2^{B-1} - 1 = 2^7 - 1 = 127$. The sample autocorrelation was computed by the program segment:

```
double *y, *R;
y = (double *) calloc(N, sizeof(double));          N-dimensional
R = (double *) calloc(M+1, sizeof(double));        (M+1)-dimensional

for (n=0; n<N; n++)                                use N=2000
       y[n] = ran1f(B, u, q, &iseed);              zero-mean y's

corr(N, y, y, M, R);                               use M=150
```

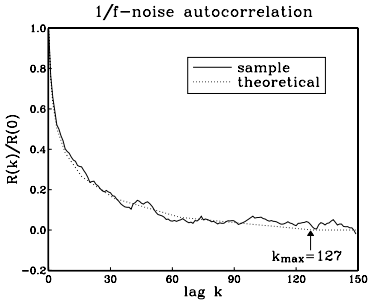


Fig. P14.5 Theoretical and sample autocorrelations for $B = 8$.

The reason for using $N = 2000$ was because we wanted to evaluate the sample autocorrelation at $M = 150$ lags. Thus, the number of lags is $150/2000 \equiv 7.5\%$ of the block length, which falls within the recommended range for getting statistically reliable lags using the routine `corr`, as was mentioned in Section A.1.

Problem B.8

We look at the cases $\omega_{\min} = 0.001\pi$ and $\omega_{\max} = 0.1\pi, 0.3\pi, 0.6\pi$. The designed model parameters are:
For $\omega_{\max} = 0.1\pi$, we have $B = 3$ and

$$c = 10, \quad \mathbf{a} = [0.6858, 0.9686, 0.9969]$$

For $\omega_{\max} = 0.3\pi$, we have $B = 4$ and

$$c = 6.6943, \quad \mathbf{a} = [0.0575, 0.8592, 0.9790, 0.9969]$$

For $\omega_{\max} = 0.6\pi$, we have $B = 6$ and

$$c = 3.5944, \quad \mathbf{a} = [-0.8850, 0.4756, 0.8541, 0.9594, 0.9887, 0.9969]$$

The spectra are shown in Figs. P14.6—P14.8.

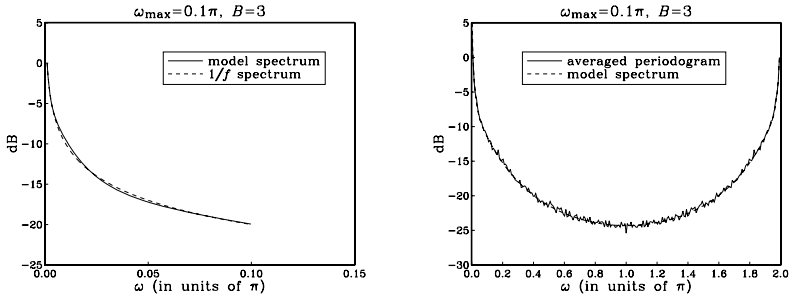


Fig. P14.6 Model spectrum and periodogram estimate. $\omega_{\max} = 0.1\pi$.

Problem B.9

Solving the equation $(0.99574)^c = 0.94791$, we find $c = 12.5309$. Then, the filter of Eq. (B.30) becomes:

$$H(z) = G\frac{(1 - 0.98444z^{-1})(1 - 0.82159z^{-1})(1 - 0.08522z^{-1})}{(1 - 0.99574z^{-1})(1 - 0.94791z^{-1})(1 - 0.51153z^{-1})}$$

The $NRR = \mathbf{h}^T\mathbf{h}$ of the filter (B.29) without the factor G can be computed analytically or much faster using MATLAB to compute a fairly long portion of the impulse response $\mathbf{h}$ (e.g., of length 2000) and then setting $G = 1/\sqrt{NRR} = 1/\sqrt{\mathbf{h}^T\mathbf{h}}$. The magnitude response squared of Eqs. (B.29) and (B.30) are shown in Fig. P14.9. The cascade realization is shown in Fig. P14.10. The corresponding sample processing algorithm will be:

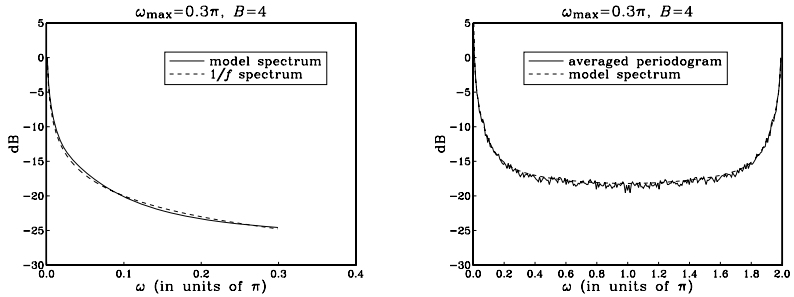


Fig. P14.7 Model spectrum and periodogram estimate. $\omega_{\max} = 0.3\pi$.

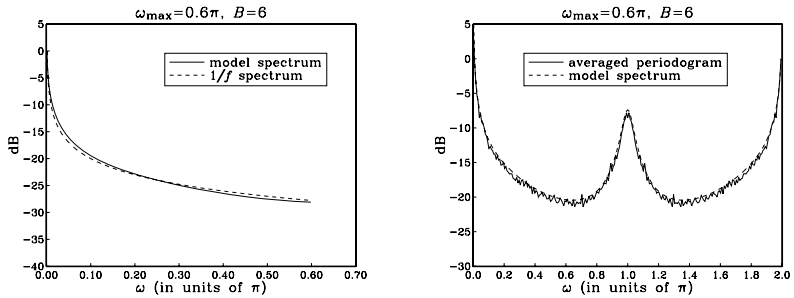


Fig. P14.8 Model spectrum and periodogram estimate. $\omega_{\max} = 0.6\pi$.

for each white noise input sample x do:

$u_0 = 0.57534\,x + 0.99574\,u_1$
$x_1 = u_0 - 0.98444\,u_1$
$u_1 = u_0$
$v_0 = x_1 + 0.94791\,v_1$
$x_2 = v_0 - 0.83392\,v_1$
$v_1 = v_0$
$w_0 = x_2 + 0.53568\,w_1$
$y = w_0 - 0.07568\,w_1$
$w_1 = w_0$

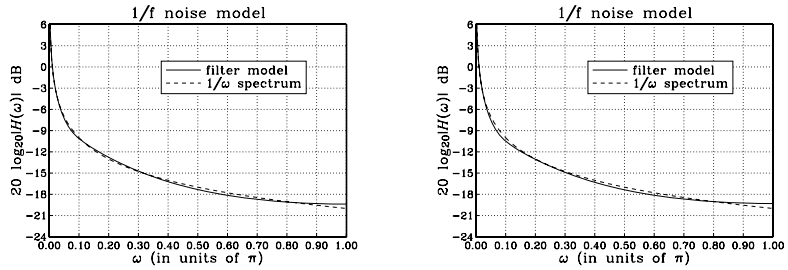


Fig. P14.9 $1/f$ noise filters for Problem B.9.

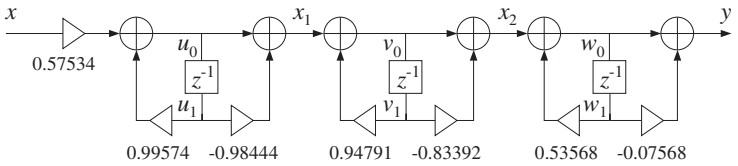


Fig. P14.10 Cascade realization of $1/f$ noise filter.