Academic year
2025-2026

Faculty of Applied Engineering

Digital Signal Processing

Signal Processing Systems – Solutions to the Exercises

Walter Daems

Master of Science in Electronics and ICT Engineering Technology
Master of Science in de industriële wetenschappen: elektronica-ICT

2210FTIESY I-Electronic Systems
2212FTIESY I-Elektronische Systemen

This document has been typeset using LaTeX and the `uantwerpendocs` package.
Calculations have been performed using Matlab/Octave.
Graphics have been composed using PGF, TiKZ and InkScape.
All this material has been prepared on a GNU/Linux workstation.

Typesetting of this document was enabled by:

DSP-SPS-2025-3.11

Contents

Chapter 2

Systems

Solution 2.3.3-1: Let's start by checking the homogeneity and then check the superposition.

Homogeniety We know that for input signal $x_1(t)$, the output of the system is $y_1(t) = \frac{5}{x_1(t)}$. The question we need to answer is whether the following relationship holds for any $a \in \mathbb{R}$:

$$x_2(t) = ax_1(t) \xrightarrow{H} y_2(t) = ay_1(t) \quad ?$$

Given the system's definition, we know:

$$\begin{aligned} y_2(t) &= \frac{5}{x_2(t)} \\ &\downarrow x_2(t) = ax_1(t) \\ &= \frac{5}{ax_1(t)} \\ &\downarrow \frac{5}{x_1(t)} = y_1(t) \\ &= \frac{y_1(t)}{a} \end{aligned}$$

The latter shows that for every $a \neq 1$ the answer to our question is no. Therefore, the system is not homogeneous and hence not linear. However, just for the sport, let's check the superposition.

Superposition We know that for input signals $x_i(t)$ with $i = 1, 2$ the corresponding output signals are: $y_i(t) = \frac{5}{x_i(t)}$ with $i = 1, 2$. The question we need to anwer is:

$$x_1(t) + x_2(t) \xrightarrow{H} y_1(t) + y_2(t) \quad ?$$

Given the system's definition, we know:

$$\begin{aligned} y(t) &= \frac{5}{x(t)} \\ &\downarrow x(t) = x_1(t) + x_2(t) \\ &= \frac{5}{x_1(t) + x_2(t)} \\ &\neq \frac{5}{x_1(t)} + \frac{5}{x_2(t)} = y_1(t) + y_2(t) \end{aligned}$$

The latter shows that the answer to our question is no. Therefore again, the system is not linear.

Solution 2.3.3-2: Let's start by checking the homogeneity and then check the superposition.

Homogeniety We know that for input signal $x_1[n]$, the output of the system is $y_1[n] = 5\sin(x_1[n])$. The question we need to answer is whether the following relationship holds for any $a \in \mathbb{R}$:

$$x_2[n] = ax_1[n] \xrightarrow{H} y_2[n] = ay_1[n] \quad ?$$

Given the system's definition, we know:

$$\begin{aligned} y_2[n] &= 5\sin(x_2[n]) \\ &\downarrow x_2[n] = ax_1[n] \\ &= 5\sin(ax_1[n]) \\ &\neq a5\sin(x_1[n]) = ay_1[n] \end{aligned}$$

The latter shows that for every $a \neq 1$ the answer to our question is no. Therefore, the system is not homogeneous and hence not linear.[1] However, just for the sport, let's check the superposition.

Superposition We know that for input signals $x_i[n]$ with $i = 1, 2$ the corresponding output signals are: $y_i[n] = 5\sin(x_i[n])$ with $i = 1, 2$. The question we need to anwer is:

$$x_1[n] + x_2[n] \xrightarrow{H} y_1[n] + y_2[n] \quad ?$$

Given the system's definition, we know:

$$\begin{aligned} y[n] &= 5\sin(x[n]) \\ &\downarrow x[n] = x_1[n] + x_2[n] \\ &= 5\sin(x_1[n] + x_2[n]) \\ &= 5(\sin(x_1[n])\cos(x_2[n]) + \cos(x_1[n])\sin(x_2[n])) \\ &\neq 5\sin(x_1[n]) + 5\sin(x_2[n]) = y_1[n] + y_2[n] \end{aligned}$$

The latter shows that the answer to our question is no. Therefore again, the system is not linear.

Solution 2.3.3-3: Instead of checking linearity by checking homogeneity and superposition, we can try to check linearity as a whole.
We know that for input signals $x_i[n]$ with $i = 1, 2$ the corresponding output signals are: $y_i[n] = 3nx_i[n]$ with $i = 1, 2$. The question we need to anwer is whether for every set of values $(a_1, a_2) \in \mathbb{R}^2$, the following relationship holds:

$$a_1x_1[n] + a_2x_2[n] \xrightarrow{H} a_1y_1[n] + a_2y_2[n] \quad ?$$

Given the system's definition, we know:

$$\begin{aligned} y[n] &= 3nx[n] \\ &\downarrow x[n] = a_1x_1[n] + a_2x_2[n] \\ &= 3n(a_1x_1[n] + a_2x_2[n]) \\ &= a_1 3nx_1[n] + a_2 3nx_2[n] \\ &\downarrow 3nx_i[n] = y_i[n], \quad i = 1, 2 \\ &= a_1y_1[n] + a_2y_2[n] \end{aligned}$$

Therefore the system is linear.

Solution 2.3.3-4: Instead of checking linearity by checking homogeneity and superposition, we can try to check linearity as a whole.
Though strictly not required, we can rewrite our system's description as:

$$y[n] - 0.5y[n-1] = 3x[n] - 2x[n-1]$$

[1] If the range of x_1 would be very small around 0, one could approximate $\sin(x[n]) \approx x[n]$ and one could consider the system linear.

Let's assume 2 arbitrary inputs $x_1[n]$ and $x_2[n]$ together with their corresponding outputs $y_1[n]$ and $y_2[n]$:

$$x_1[n] \xrightarrow{H} y_1[n] \qquad\qquad x_2[n] \xrightarrow{H} y_2[n]$$

This implies:

$$y_1[n] - 0.5y_1[n-1] = 3x_1[n] - 2x_1[n-1] \tag{2.1}$$
$$y_2[n] - 0.5y_2[n-1] = 3x_2[n] - 2x_2[n-1] \tag{2.2}$$

The key questions is whether:

$$ax_1[n] + bx_2[n] \xrightarrow{H} ay_1[n] + by_2[n]$$

Equivalently, does the following relationship hold?

$$(ay_1[n] + by_2[n]) - 0.5(ay_1[n-1] + by_2[n-1]) = 3(ax_1[n] + bx_2[n]) - 2(ax_1[n-1] + bx_2[n-1]) \tag{2.3}$$

Indeed, this holds, because if both (2.1) and (2.2) hold, then also any linear combination of those equations must hold, i.e.

$$a(y_1[n] - 0.5y_1[n-1]) + b(y_2[n] - 0.5y_2[n-1]) = a(3x_1[n] - 2x_1[n-1]) + b(3x_2[n] - 2x_2[n-1])$$

which can easily be reworked to (2.3).

Solution 2.3.4-1: The question to answer is

$$x(t) \xrightarrow{H} y(t) \overset{?}{\Longrightarrow} x(t-\tau) \xrightarrow{H} y(t-\tau)$$

for an arbitrary value of τ.
Start by noting that the $x(t)$ and $y(t)$ shifted over an amount τ (to the right) are:

$$x(t-\tau)$$
$$y(t-\tau) = \frac{5}{x(t-\tau)}$$

The procedure is to take the system description and shift the input signal over an amount *tau* (to the right) and see whether the output equals $y(t-\tau)$.
In this way,

$$x(t-\tau) \xrightarrow{H} \frac{5}{x(t-\tau)} = y(t-\tau)$$

And therefore, the system is time invariant.

Solution 2.3.4-2: The question to answer is

$$x[n] \xrightarrow{H} y[n] \overset{?}{\Longrightarrow} x[n-n_0] \xrightarrow{H} y[n-n_0]$$

for an arbitrary value of n_0.
Start by noting that the $x(t)$ and $y(t)$ shifted over an amount τ (to the right) are:

$$x[n-n_0]$$
$$y[n-n_0] = 5\sin(x[n-n_0])$$

The procedure is to take the system description and shift the input signal over an amount n_0 (to the right) and see whether the output equals $y[n-n_0]$.
In this way,

$$x[n-n_0] \xrightarrow{H} 5\sin(x[n-n_0]) = y[n-n_0]$$

And therefore, the system is time invariant.

Solution 2.3.4-3: The question to answer is

$$x[n] \xrightarrow{H} y[n] \stackrel{?}{\Longrightarrow} x[n-n_0] \xrightarrow{H} y[n-n_0]$$

for an arbitrary value of n_0.
Start by noting that the $x(t)$ and $y(t)$ shifted over an amount τ (to the right) are:

$$\begin{aligned} &x[n-n_0] \\ &y[n-n_0] = 3(n-n_0)x[n-n_0] \end{aligned}$$

The procedure is to take the system description and shift the input signal over an amount n_0 (to the right) and see whether the output equals $y[n-n_0]$.
In this way,

$$x[n-n_0] \xrightarrow{H} 3nx[n-n_0] \neq y[n-n_0] = 3(n-n_0)x[n-n_0]$$

And therefore, the system is *not* time invariant.

Solution 2.3.4-4: We can rewrite the system's description as:

$$x[n] \xrightarrow{H} y[n] \text{ with } y[n] - 0.5y[n-1] = 3x[n] - 2x[n-1]$$

We'll call this the *alternate* system description. The question to answer is

$$x[n] \xrightarrow{H} y[n] \stackrel{?}{\Longrightarrow} x[n-n_0] \xrightarrow{H} y[n-n_0]$$

for an arbitrary value of n_0.
Start by noting that the $x(t)$ and $y(t)$ shifted over an amount τ (to the right) are:

$$\begin{aligned} &x[n-n_0] \\ &y[n-n_0] = 0.5y[n-n_0-1] + 3x[n-n_0] - 2x[n-n_0-1] \end{aligned}$$

The last equation can be rewritten as:

$$y[n-n_0] - 0.5y[n-n_0-1] = 3x[n-n_0] - 2x[n-n_0-1]$$

The procedure is to take the *alternate* system description and shift the input signal over an amount n_0 (to the right) and see whether the candidate output $y[n-n_0]$ still fulfills the system description.
In this way,

$$\begin{aligned} x[n-n_0] \xrightarrow{H} &u[n] \text{ with } u[n] - 0.5u[n-1] = 3x[n-n_0] - 2x[n-n_0-1] \\ &\downarrow\ 3x[n-n_0] - 2x[n-n_0-1] = y[n-n_0] - 0.5y[n-n_0-1] \\ \xrightarrow{H} &u[n] \text{ with } u[n] - 0.5u[n-1] = y[n-n_0] - 0.5y[n-n_0-1] \end{aligned}$$

Again, as $u[n]$ is the solution to a difference equation, we can pick an integration constant. If we pick it such that for an arbitrary timepoint k, $u[k-1] = y[k-n_0-1]$, then we we can use induction to prove $u[n] = y[n-n_0]$ for any n, and therefore the system is time invariant.

Solution 2.5.1-1: As the system description is an equation, it can always be reworked. A good first step is to solve it for y with the highest argument, in this case $y[n]$. Therefore no rework is done. It is already in the correct form.

Now, without loss of generality we assume $y[n \to -\infty] = 0$. In addition, as we desire to determine the impulse response in our experiment, we know that $x[n] = 0$ for negative times. As a consequence, also $y[n] = 0$ for negative times.

Conclusion: the system is causal.

Is our assumption really without loss of generality? Yes, if $y[n]$ would already be active for negative times, then our system would only show new behavior because of the impulse at the input when $n = 0$. Therefore, it is causal.

Solution 2.5.1-2: As the system description is an equation, it can always be reworked. A good first step is to solve it for y with the highest argument, in this cas $y[n]$. Therefore no rework is done. It is already in the correct form.

If $x[n] = 0$ for $n < 0$, then also $y[n] = 0$ for $n \leq 0$. Therefore, the system is causal.

Solution 2.5.1-3: At first glance, to calculate $y[n]$ we need to have $y[n+1]$ available, which is information from the future.

However, the system description is an equation, and therefore it can always be reworked. A good first step is to solve it for y with the highest argument, in this cas $y[n+1]$. This results in:

$$x[n] \xrightarrow{H} y[n+1] = \frac{1}{1.4}\left(2x[n] - y[n]\right)$$

Now, without loss of generality we assume $y[n \to -\infty] = 0$. In addition, as in our experiment we desire to determine the impulse response, we know that $x[n] = 0$ for negative times. As a consequence, also $y[n] = 0$ for negative times.

The reaction to the impulse will only occur at $n = 1$. Conclusion: the system is causal.

Solution 2.5.1-4: Again, the first step is to solve the system description for y with the highest argument, i.e. $y[n]$ in this case:

$$\begin{aligned} x[n] &\xrightarrow{H} y[n] - 0.5y[n] = -1.4y[n-1] + 2x[n+1] \\ &\xrightarrow{H} y[n] = -2.8y[n-1] + 4x[n+1] \end{aligned}$$

Now, without loss of generality let's assume $y[n \to -\infty] = 0$. In addition, as in our experiment we desire to determine the impulse response, we know that $x[n] = 0$ for negative times. As a consequence, also $y[n] = 0$ for $n < -1$. Indeed at $n = -1$ the equation tells us that $y[-1] = 4$ as $x[0] = 1$ in an impulse.

Therefore the system is not causal.

Solution 2.5.1-5: If apply the inverse Z-transform to the transfer function, we obtain the impulse response of the system:

$$\delta[n] \xrightarrow{H} [\underbrace{1}_{n=0}, -0.2, 4.3, 0, 0, \ldots]$$

Therefore, the system is causal.

Solution 2.5.1-6: If apply the inverse Z-transform to the transfer function, we obtain the impulse response of the system:

$$\delta[n] \xrightarrow{H} 1, 0.34, \underbrace{-1}_{n=0}, -0.2, 4.3, 0, 0, \ldots$$

As this system responds for times ($n = -2$ and $n = -1$) prior to the arrival of the impulse at $n = 0$, it is *not* causal.

Solution 2.5.1-7: We need to calculate the inverse Z-transform of the transfer function. We take one of the possible routes.

$$\begin{aligned} H(z) &= \frac{z^2 + 3z + 1}{z - 1} \\ &\downarrow \text{Long division} \\ &= z + 4 + 5z^{-1} + \ldots \end{aligned}$$

And therefore the inverse Z-transform results in the following impulse response:

$$\delta[n] \xrightarrow{H} [1, \underbrace{4}_{n=0}, 5, \ldots]$$

As this system responds for time $n = -1$ prior to the arrival of the impulse at $n = 0$, it is *not* causal. Note that the long division will always result in a term with positive power of z when the degree of the numerator is higher than the degree of the denominator.

Solution 2.5.1-8: We need to calculate the inverse Z-transform of the transfer function. We take one of the possible routes.

$$\begin{aligned} H(z) &= \frac{z^2 + 3z + 1}{3.2z^2 - 1} \\ &\downarrow \text{Long division} \\ &= \frac{1}{3.2} + \frac{3}{3.2}z^{-1} + \ldots \end{aligned}$$

And therefore the inverse Z-transform results in the following impulse response:

$$\delta[n] \xrightarrow{H} [\underbrace{\frac{1}{3.2}}_{n=0}, \frac{3}{3.2}, \ldots]$$

As this system doesn't start responding before the arrival of the impulse arriving at $n = 0$, it is causal. Note that the long division is a bit overkill. Inspecting the degrees of numerator and denominator would suffice. If the degree of the numerator is lower or equal than the degree of the denominator, the long division will not result in terms with positive powers of z and the system is causal.

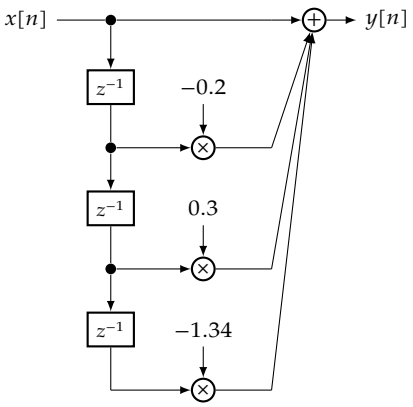
Solution 2.5.1-9: The system is causal as the degree of the denominator is higher or equal to the degree of the numerator. If you don't understand this reasoning, solve the previous exercises.

Solution 2.5.1-10: Apparently, we need to know the output of the system at $n = 0$ to know the output at $n = 0$, so the system doesn't seem to be causal. However, don't let yourself be fooled. Simple rearrangement of the system's description leads to:

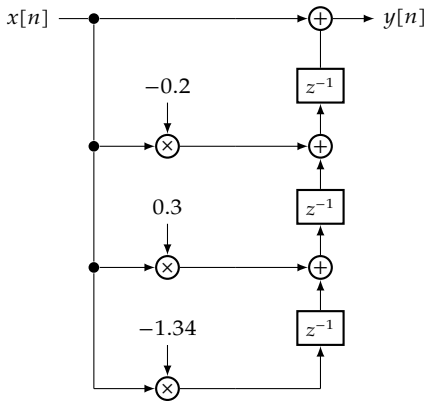
$$x[n] \xrightarrow{H} y[n] = \frac{1}{1.1}y[n-1] - \frac{0.5}{1.1}x[n-2]$$

The impulse response of this system only starts at $n = 2$, and therefore, the system is causal.

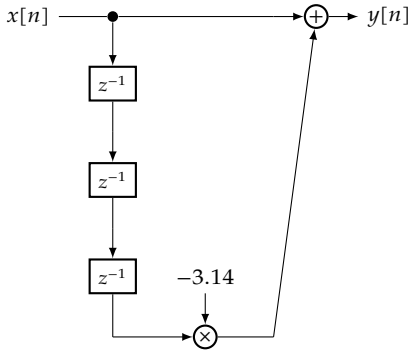
Solution 2.8.1.3-1: Direct form implementation:



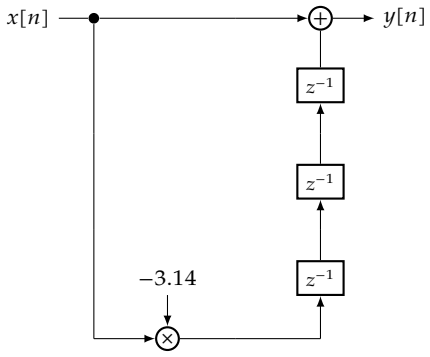
The transposition procedure yields the transposed form:



Solution 2.8.1.3-2: Direct form implementation:

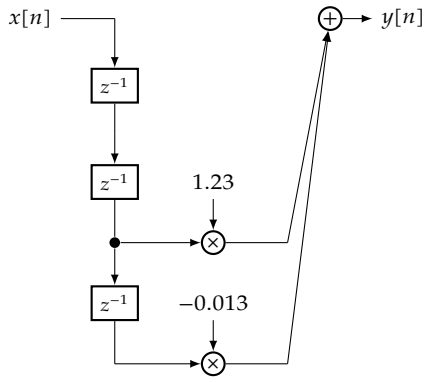


The transposition procedure yields the transposed form:

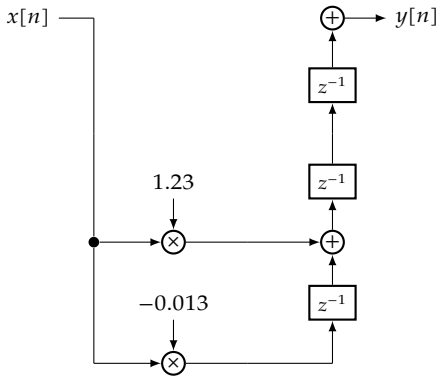


As you can see, the difference is rather trivial in this case.

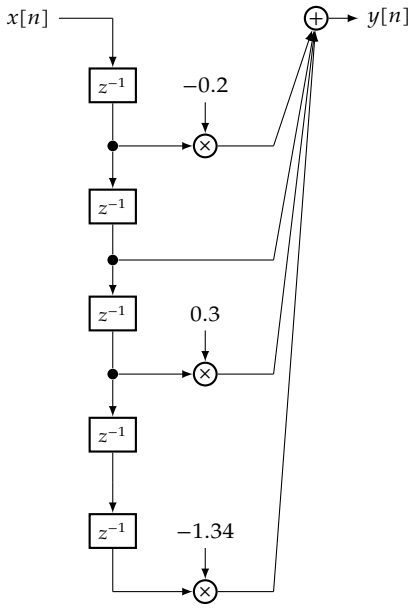
Solution 2.8.1.3-3: Direct form implementation:



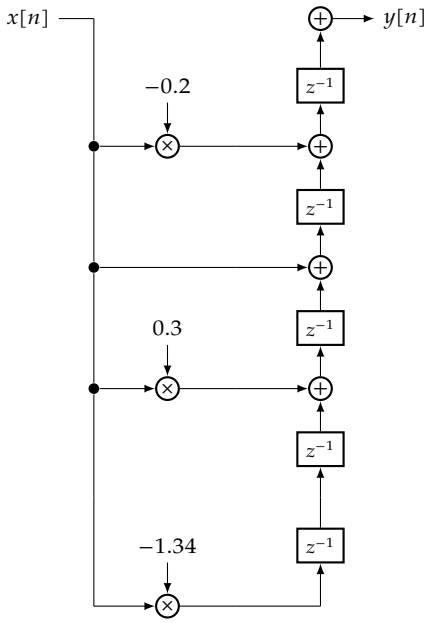
The transposition procedure yields the transposed form:



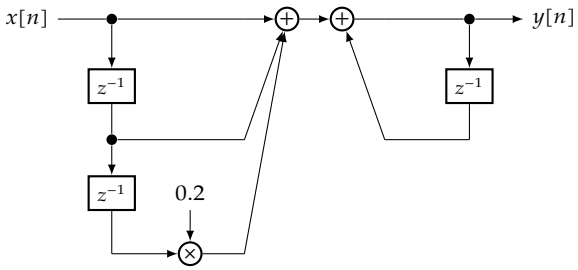
Solution 2.8.1.3-4: Direct form implementation:



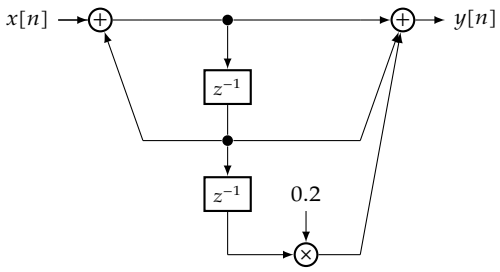
The transposition procedure yields the transposed form:



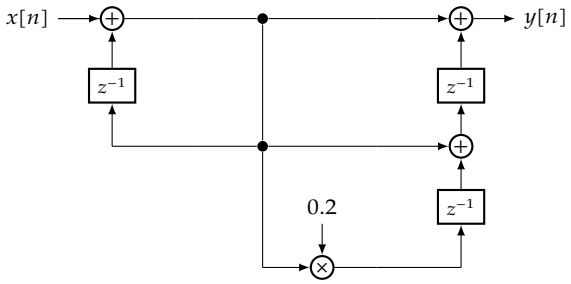
Solution 2.8.2.5-1: The direct form I implementation:



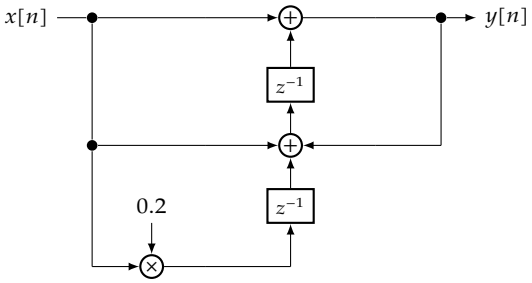
The direct form II implementation:



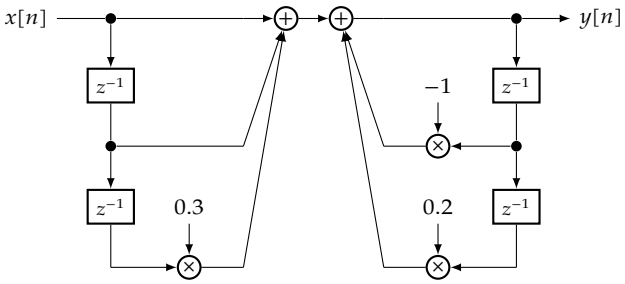
The transposed form I implementation:



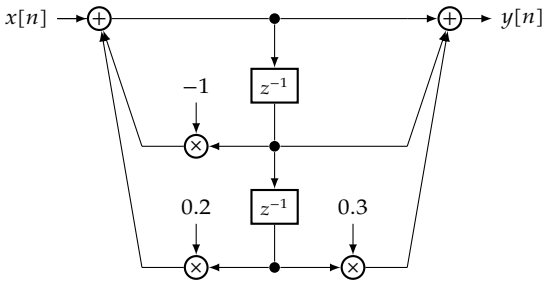
The transposed form II implementation:



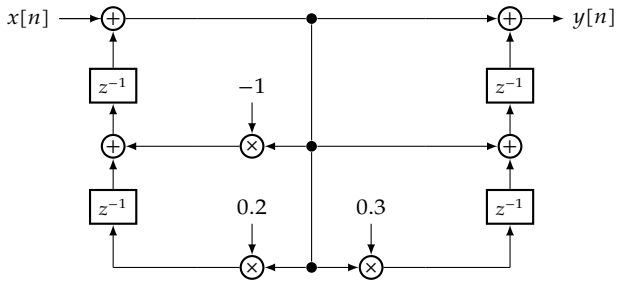
Solution 2.8.2.5-2: The direct form I implementation:



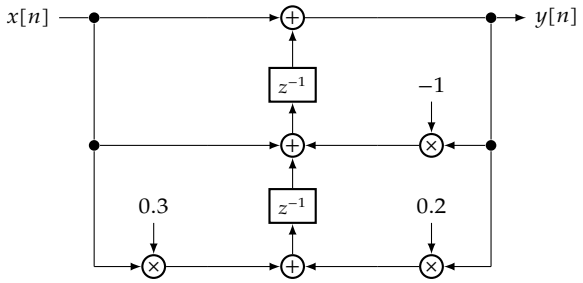
The direct form II implementation:



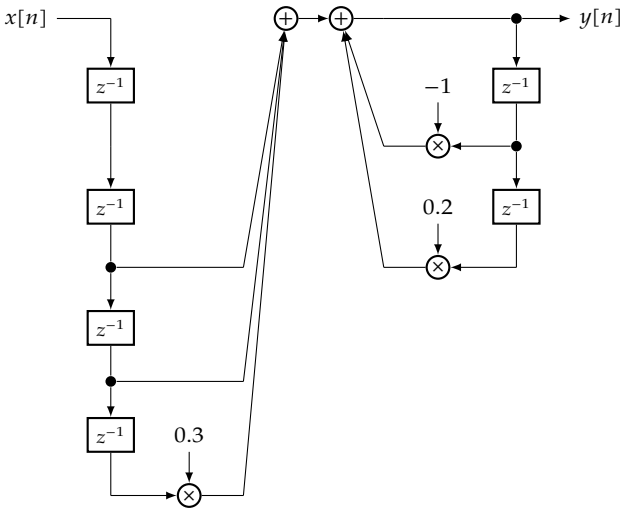
The transposed form I implementation:



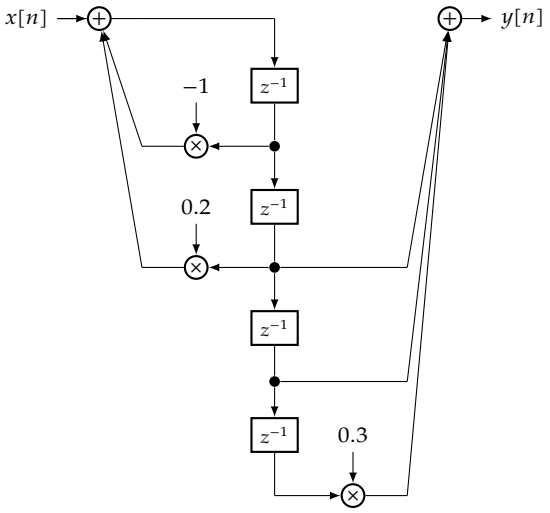
The transposed form II implementation:



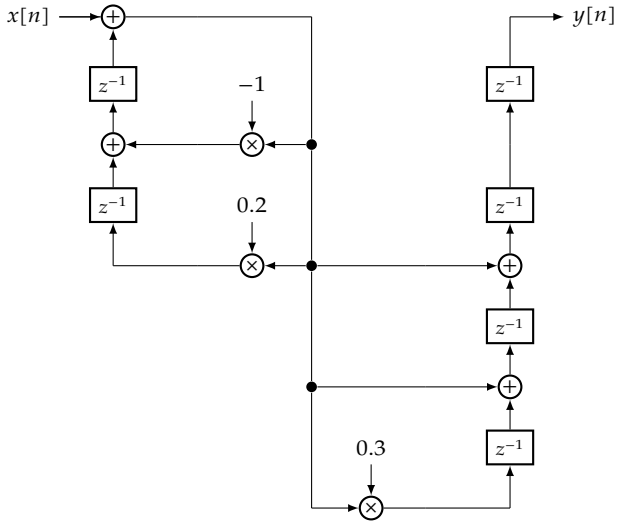
Solution 2.8.2.5-3: The direct form I implementation:



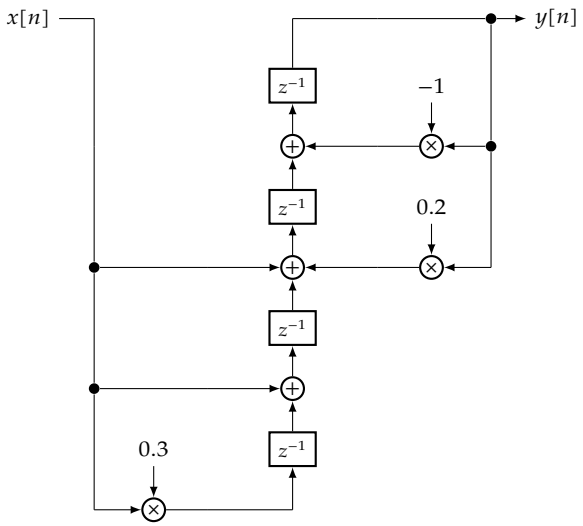
The direct form II implementation:



The transposed form I implementation:



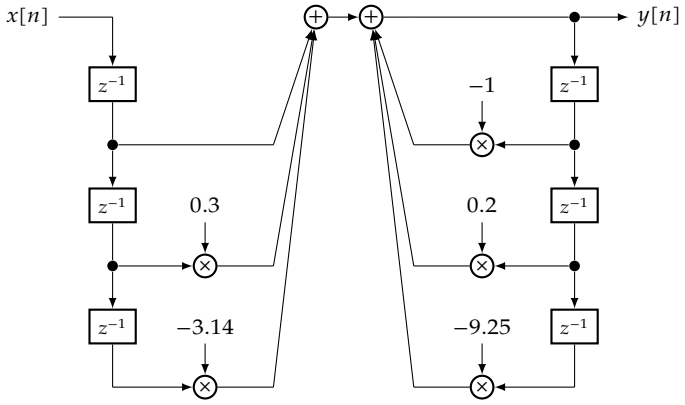
The transposed form II implementation:



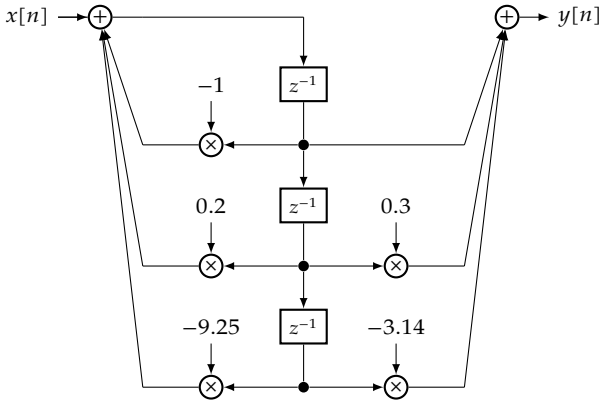
Solution 2.8.2.5-4: The first step is to divide the numerator and denominator by z^3:

$$H(z) = \frac{z^{-1} + 0.3z^{-2} - 3.14z^{-3}}{1 + z^{-1} - 0.2z^{-2} + 9.25z^{-3}}$$

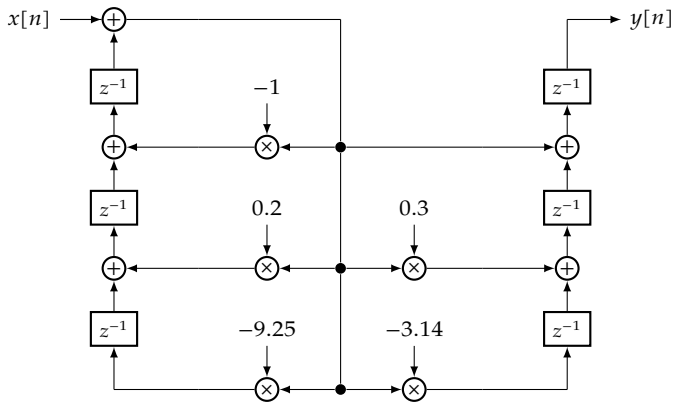
The subsequent implementation are straightforward.
The direct form I implementation:



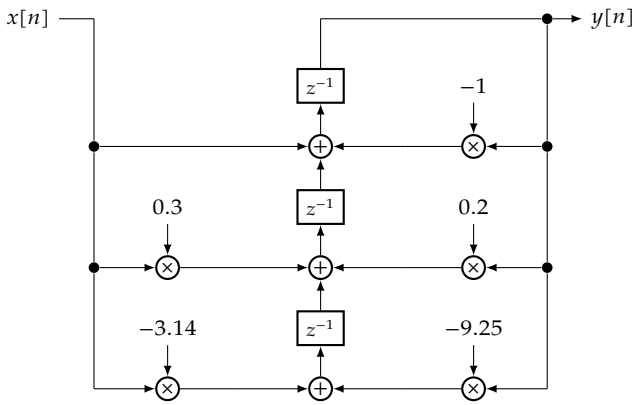
The direct form II implementation:



The transposed form I implementation:



The transposed form II implementation:



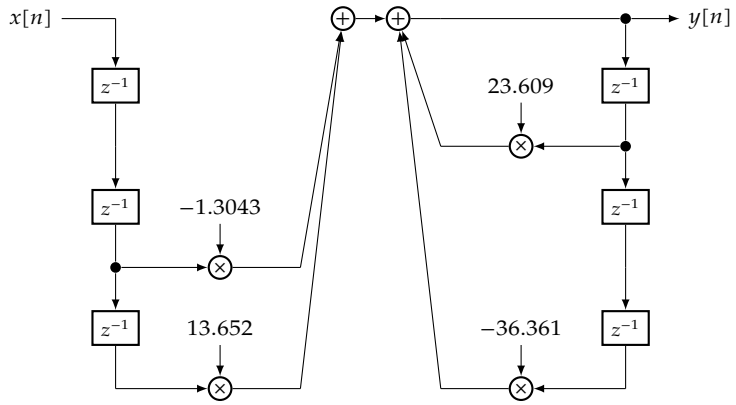
Solution 2.8.2.5-5: The first step is to normalize the transfer function:

$$H(z) = \frac{-1.3043z + 13.652}{z^3 - 23.609z^2 + 36.361}$$

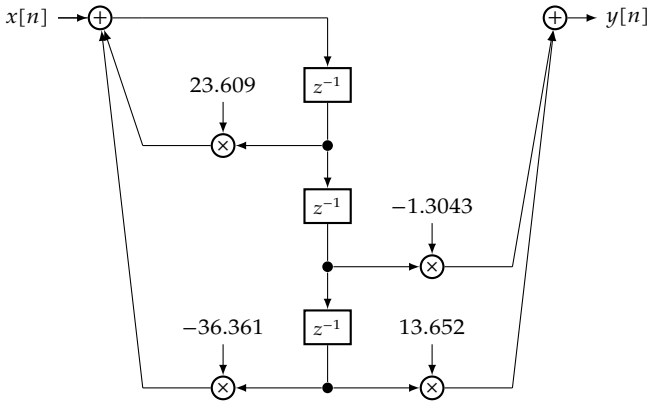
and then divide numerator and denominator by z^3:

$$H(z) = \frac{-1.3043z^{-2} + 13.652z^{-3}}{1 - 23.609z^{-1} + 36.361z^{-3}}$$

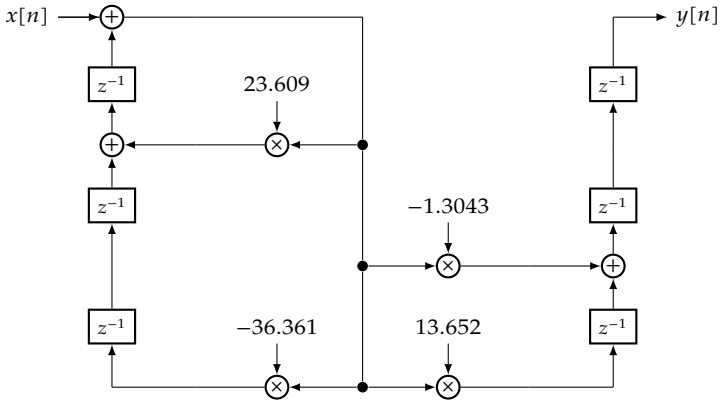
The direct form I implementation:



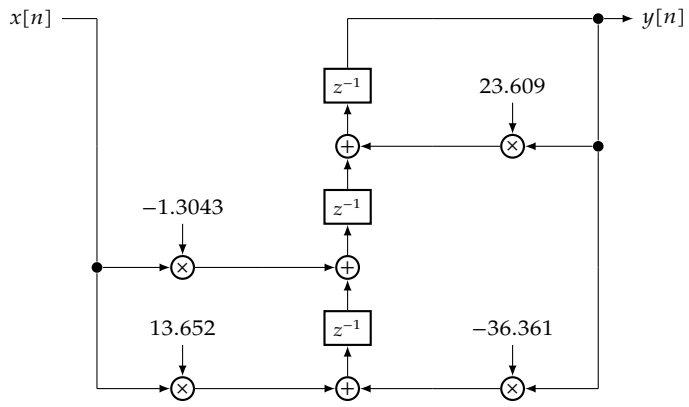
The direct form II implementation:



The transposed form I implementation:



The transposed form II implementation:



Solution 2.8.2.5-6: See the starting equation of the exercise whose solution you are trying to reverse engineer.

Solution 2.8.2.5-7: See the starting equation of the exercise whose solution you are trying to reverse engineer.

Chapter 3

System architectures

Solution 3.3.4-1: Let's start with the **single-buffer** setup: Given a quad-core, operations 1 and 2 can run in parallel, but not together with operatoin 3. Therefore:

$$\begin{aligned} CPLF &= \frac{f_s}{M}\left(\max(T_1, T_2) + T_3\right) \\ &= \frac{750\,\text{kHz}}{128}\left(120\,\mu\text{s} + 75\,\mu\text{s}\right) \\ &= 1.14 \end{aligned}$$

The conclusion is, that the hardware cannot handle this, as $T > 1$. The maximal achievable sampling frequency is (just invert the previous equation):

$$\begin{aligned} f_s &= \frac{M}{\max(T_1, T_2) + T_3} \\ &= \frac{128}{120\,\mu\text{s} + 75\,\mu\text{s}} \\ &= 656\,\text{kHz} \end{aligned}$$

The memory requirement is easily determined to be:

$$R = 3M + \sum_{i=1}^{V}(M + R_{i,extra}) = 3\dot{1}28 + (128 + 16) + (128 + 0) + (128 + 32) = 816$$

The latency in the original case amounts to:

$$L = 2\frac{M}{f_s} = 2\frac{128}{750\,\text{kHz}} = 341\,\mu\text{s}$$

Of course, it will be higher for a feasible sample frequency.

Then let's consider the **double-buffer** setup: Given a quad-core, operations 1, 2 and 3 can run in parallel. Therefore:

$$\begin{aligned} CPLF &= \frac{f_s}{M}\max(T_1, T_2, T_3) \\ &= \frac{750\,\text{kHz}}{128} \cdot 120\,\mu\text{s} \\ &= 0.7 \end{aligned}$$

The conclusion is, that the hardware can handle this, as $T \ll 1$. The maximal achievable sampling frequency is (just invert the previous equation):

$$\begin{aligned} f_s &= \frac{M}{\max(T_1, T_2, T_3)} \\ &= \frac{128}{120\,\mu\text{s}} \\ &= 1.06\,\text{MHz} \end{aligned}$$

The memory requirement is easily determined to be:

$$R = 2M + \sum_{i=1}^{V}(2M + R_{i,extra}) = 2\dot{1}28 + (2 \cdot 128 + 16) + (2 \cdot 128 + 0) + (2 \cdot 128 + 32) = 1072$$

The latency in the original case amounts to:

$$L = (1 + 2)\frac{M}{f_s} = 3\frac{128}{750\,\text{kHz}} = 512\,\mu\text{s}$$

Chapter 4

Convolution and FFT-convolution

Solution 4.3.3-1: (a) Using the convolution's definition:
We start with the definition of the convolution operation:

$$\begin{aligned} x[n]\star y[n] &= \sum_{i=-\infty}^{+\infty} x[n-i]y[i] \\ &\downarrow \text{variable substitution } k = n-i \\ &= \sum_{k=-\infty}^{+\infty} x[k]y[n-k] = y[n]\star x[n] \end{aligned}$$

(b) Using the Z-transform:

$$\begin{aligned} x[n]\star y[n] &\xrightarrow{Z} X(z).Y(z) \\ &\downarrow \text{compex multiplication is commutative} \\ &\xrightarrow{Z} Y(z).X(z) \xrightarrow{Z^{-1}} y[n]\star x[n] \end{aligned}$$

Solution 4.3.3-2: (a) Using the convolution's definition:
We start with the definition of the convolution operation:

$$\begin{aligned} (x[n]\star y[n])\star z[n] &= \left(\sum_{i=-\infty}^{+\infty} x[n-i]y[i]\right)\star z[n] \\ &= \sum_{k=-\infty}^{+\infty}\left(\sum_{i=-\infty}^{+\infty} x[n-k-i]y[i]\right)z[k] \\ &= \sum_{k=-\infty}^{+\infty}\sum_{i=-\infty}^{+\infty} x[n-k-i]y[i]z[k] \\ &\downarrow k+i=m \longleftrightarrow i=m-k \\ &= \sum_{k=-\infty}^{+\infty}\sum_{m=-\infty}^{+\infty} x[n-m]y[m-k]z[k] \\ &= \sum_{m=-\infty}^{+\infty} x[n-m]\sum_{k=-\infty}^{+\infty} y[m-k]z[k] \\ &= \sum_{m=-\infty}^{+\infty} x[n-m]\,(y[m]\star z[m]) \\ &= x[n]\star(y[n]\star z[n]) \end{aligned}$$

(b) Using the Z-transform:

$$\begin{aligned} (x[n]\star y[n])\star z[n] &\xrightarrow{Z} (X(z)Y(z))\,Z[z] \\ &\downarrow \text{compex multiplication is associative} \\ &\xrightarrow{Z} X(z)\,(Y(z)Z(z)) \xrightarrow{Z^{-1}} x[n]\star(y[n]\star z[n]) \end{aligned}$$

Solution 4.3.3-3: (a) Using the convolution's definition:
We start with the definition of the convolution operation:

$$\begin{aligned}(x[n]\star(y[n]+z[n]) &= \sum_{i=-\infty}^{+\infty} x[n-i](y[i]+z[n]) \\ &= \sum_{i=-\infty}^{+\infty} (x[n-i]y[i]+y[n-i]z[n]) \\ &= \sum_{i=-\infty}^{+\infty} x[n-i]y[i] + \sum_{i=-\infty}^{+\infty} y[n-i]z[n] \\ &= x[n]\star y[n] + x[n]\star z[n]\end{aligned}$$

(b) Using the Z-transform:

$[n]\star(y[n]+z[n]) \xrightarrow{Z} X(z)\cdot(Y(z)+Z(z))$ ↓ complex multiplication is distributive w.r.t. complex additio

$\xrightarrow{Z} X(z)Y(z)+X(z)Z(z) \xrightarrow{Z^{-1}} x[n]\star y[n]+x[n]\star z[n]$

Listing 4.1: Incremental input-side convolution algorithm using normal pointers

Solution 4.7.1-1:

```
double iconvinc_p( double           x,  // next input sample
                   double*          h,  // impulse response array
                   double*          z,  // circular buffer array
                   double*&         zi, // circular buffer array pointer
                   int              n ) // impulse reponse length
{
  double*const endh = h + n;
  double*const endz = z + n;
  for ( ; h < endh; ++h, ++zi )
    {
      if( zi >= endz )
        zi = z;
      *zi += x * *h;                      // calculate x[i].h[n] (MAC)
    }
  double y = *zi;

  *zi = 0;                              // circular pointer increment
  if ( ++zi >= endz )
    zi = z;

  return y;
}
```

Solution 4.7.1-2: Submit your solution to your instructor. He will be happy to check your result.

Solution 4.7.1-3: Submit your solution to your instructor. He will be happy to check your result.

Listing 4.2: Incremental output-side convolution algorithm using pointers

Solution 4.7.1-4:

```
double oconvinc_p( double           x,  // next input sample
                   double*          h,  // impulse response array
                   double*          z,  // circular buffer array
                   double*&         zi, // circular pointer into
                                        //   circular buffer array
                   int              n ) // impulse reponse length
{
  *zi = x;                              // store current input sample

  double y = 0.0;
  double*const endh = h + n;
  for ( ; h < endh; ++h, --zi )
    {
      if( zi < z )
        zi += n;
      y += *zi * *h;                    // calculate output for n = i (MAC)
    }
```

```
  ++zi;                                   // modulo n index increment
  if ( zi >= z + n )
    zi = z;

  return y;
}
```

Solution 4.7.1-5: Submit your solution to your instructor. He will be happy to check your result.

Solution 4.7.1-6: Submit your solution to your instructor. He will be happy to check your result.

Solution 4.9.4-1:

Case 1 We can only afford an incremental implementation (probably in hardware), leading to:

$$R = 2N + 2 = 50 \sim 100\,\mathrm{B} \qquad L = 1$$

In this case, the total amount of work per sample and the *CPLF*make no sense.

Case 2 In this case, the overperformance graphs indicate clearly that only a direct (time-domain based) implementation makes sense. Given a single-buffer strategy, and calculating $K = M + N - 1 = 55$, this leads to:

$$R = K + N = 79 \sim 158\,\mathrm{B} \qquad L = 2M = 64\,\mathrm{samples}$$

$$T = MNT_{MAC} = 19.2\,\mu\mathrm{s} \qquad CPLF = \frac{f_s}{M}T = 6\%$$

in which we chose for the outbut-buffer style OA or the input-buffer style OS method, as they avoid additional copy operations.

Case 3 Also in this case, the overperformance graphs show no advantage in executing an FFT-convolution. Given a double-buffer strategy and calculating $K = M + N - 1 = 287$, this leads to

$$R = 2K + 1 = 575 \sim 1.150\,\mathrm{kB} \qquad L = 2M = 512\,\mathrm{samples}$$

$$T = MNT_{MAC} = 204.8\,\mu\mathrm{s} \qquad CPLF = \frac{f_s}{M}T = 8\%$$

in which we chose for the outbut-buffer style OA or the input-buffer style OS method, as they avoid additional copy operations.

Case 4 In this case, reading the overperformance graphs, we can expect a speed-up of about 5 using an indirect (frequency-domain based) implementation. Using a mixed-buffer strategy (for which the equations of the single-buffer strategy hold) and calculating $K = M + N - 1 = 1535$, we obtain:

$$R = 3K - M = 3581 \sim 7.162\,\mathrm{kB} \qquad L = 2M = 2\,048\,\mathrm{samples}$$

$$T = (K + N - 1)T_{COP} + 2K(1 + 2\log_2 K)T_{MAC} = 1.721\,9\,\mathrm{ms} \qquad CPLF = \frac{f_s}{M}T = 17\%$$

Solution 4.9.4-2: Let's answer all the questions:

1. The latency of a single-buffer system is determined by:

$$L = \frac{2M}{f_s}$$

Ensuring $L \leq 10\,\text{ms}$, this leads to:

$$M \leq \frac{10\,\text{ms} f_s}{2} = 960$$

Conclusion, we cannot use a buffer length of 1024 samples, but need to restrict ourselves to $2^9 = 512$ samples.

2. As memory seems not to be constrained, we can choose the best implementation scheme only based on speed. Given $M = 512$ and $N = 384$, we can read from the overperformance graphs a speed up of about 5 for the indirect methods. Given the fact that copy operations are cheaper than MAC-operations, choosing an *overlap-save* method is optimal.

3. The time required for a single channel single frame, is:

$$T_1 = (K + N - 1)T_{COP} + 2K(1 + 2\log_2 K)T_{MAC}$$

with $K = M + N - 1 = 895$. This leads to

$$T_1 = \left(895 \cdot 5 + 2 \cdot 985 \cdot (1 + 2\log_2 895) \cdot 10\right)\text{ns} = 375.34\,\mu\text{s}$$

This leads to a *CPLF* of

$$CPLF = \frac{f_s}{M}\left(1 - p + \frac{p}{Q}\right)T_1$$

with $p = 0.9$, $Q = 2$, $f_s = 192\,\text{kHz}$ and $M = 512$, leading to:

$$CPLF = 7.74\%$$

i.e. that the system is loaded for about 15% for the full stereo system.

4. The amount of memory required for the stereo system amounts to:

$$R = (3M + 2M + 3N - 3) \cdot 3\,\text{byte/sample} = 11.127\,\text{kB}$$

The total reverb length amounts to 384 samples, correspondig to a reverb time of:

$$T_{reverb} = \frac{N}{f_s} = 2\,\text{ms}$$

This will be hardly audible. Even for a small room, one needs a reverb time of a few 100s of milliseconds.

Solution 4.9.4-3: Submit your architecture to the lecturers. They will check your result.

Chapter 5

Digital Filtering

Solution 5.2.1.1-1: The filter is described in the time domain as:

$$y[n] = \frac{1}{N}\sum_{i=0}^{N-1} x[n-i]$$

Let's apply the Z-transform to both sides of the equation:

$$\begin{aligned}
\mathcal{Z}(y[n]) &= \mathcal{Z}\left(\frac{1}{N}\sum_{i=0}^{N-1} x[n-i]\right) \\
Y(z) &= \frac{1}{N}\sum_{i=0}^{N-1} \mathcal{Z}(x[n-i]) \\
&= \frac{1}{N}\sum_{i=0}^{N-1} X(z)z^{-i} \\
&= \frac{1}{N}X(z)\sum_{i=0}^{N-1} z^{-i}
\end{aligned}$$

And therefore:

$$\begin{aligned}
H(z) &= \frac{Y(z)}{X(z)} \\
&= \frac{1}{N}\sum_{i=0}^{N-1} z^{-i}
\end{aligned}$$

Solution 5.2.1.1-2: The impulse response of this filter is:

$$h[n] = \frac{1}{N}\cdot[\underbrace{1}_{n=0}, 1, 1, \ldots, \underbrace{1}_{n=N-1}]$$

The DtFT of this function has been elaborated in the course notes in the section on spectral leak. The result can be found there:

$$\text{DtFT}\left(h[n]\right) = \frac{1}{N}\,\mathrm{e}^{-\mathrm{j}\frac{\omega(N-1)t_S}{2}}\frac{\sin\left(\frac{\omega N T_s}{2}\right)}{\sin\left(\frac{\omega T_s}{2}\right)}$$

An alternative (equally good) elaboration is:

$$\begin{aligned}
\text{DtFT}\left(h[n]\right) &= \frac{1}{N}\sum_{n=0}^{N=1} \mathrm{e}^{-\mathrm{j}\omega n T_s} \\
&= \frac{1}{N}\,\mathrm{e}^{-\mathrm{j}\frac{\omega(N-1)T_s}{2}}\left(\mathrm{e}^{\mathrm{j}\frac{\omega(N-1)T_s}{2}} + \mathrm{e}^{\mathrm{j}\frac{\omega(N-3)T_s}{2}} + + \ldots + \mathrm{e}^{-\mathrm{j}\frac{\omega(N-3)T_s}{2}} + \mathrm{e}^{-\mathrm{j}\frac{\omega(N-1)T_s}{2}}\right)
\end{aligned}$$

In this sequence, we can combine the outer terms, the second outer terms and so on, using the well known equation:

$$\cos\alpha = \frac{\mathrm{e}^{\mathrm{j}\alpha} + \mathrm{e}^{-\mathrm{j}\alpha}}{2}$$

If N is odd, this results in:

$$\text{DtFT}\left(h[n]\right)) = \frac{1}{N}\,\mathrm{e}^{-\mathrm{j}\frac{\omega(N-1)T_s}{2}}\left(\cos\left(\frac{\omega(N-1)T_s}{2}\right)+\cos\left(\frac{\omega(N-3)T_s}{2}\right)+\ldots+\cos\left(\omega T_s\right)\right)+1\Bigg)$$

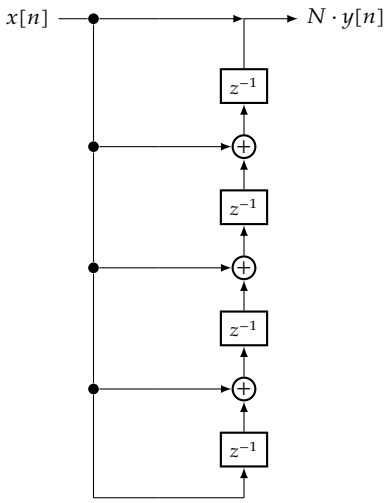
If N is even, this results in:

$$\text{DtFT}\left(h[n]\right)) = \frac{1}{N}\,\mathrm{e}^{-\mathrm{j}\frac{\omega(N-1)T_s}{2}}\left(\cos\left(\frac{\omega(N-1)T_s}{2}\right)+\cos\left(\frac{\omega(N-3)T_s}{2}\right)+\ldots+\cos\left(\frac{\omega T_s}{2})\right)\right)$$

The conclusion, is that the filter exhibits a linear phase, and except for this phase factor, has a real DtFT. Plots of this function (the so-called *Dirichlet kernel*) can be found in the section on spectral leak.

Solution 5.2.1.1-3: Let's draw a solution for $N = 5$. We omit the factor $1/N$ because it is only an overall scaling factor. The direct form is easily found to be:

The transposed form is easily found to be:



Again, we omitted the scaling factor $1/N$.

Solution 5.2.1.1-4: Submit your solution to your instructor. He will be happy to check your result.

Solution 5.2.1.2-1: The filter is described in the time domain as:

$$y[n] = y[n-1] + \frac{1}{N}\left(x[n] - x[n-N]\right)$$

Let's apply the Z-transform to both sides of the equation:

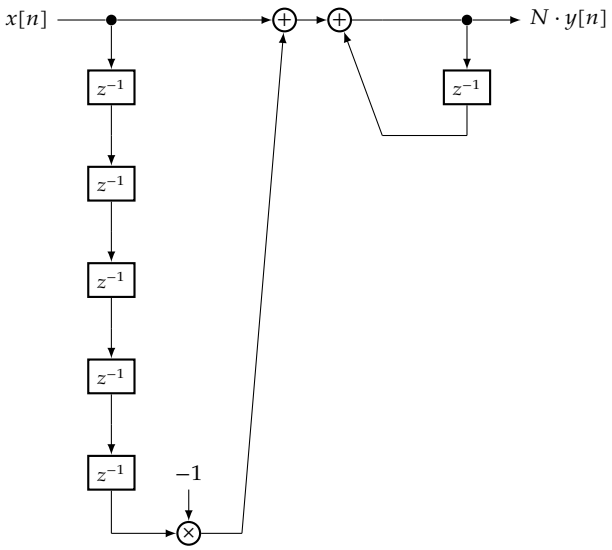
$$\begin{aligned}
\mathcal{Z}(y[n]) &= \mathcal{Z}\left(y[n-1]) + \frac{1}{N}\left(x[n] - x[n-N]\right)\right) \\
Y(z) &= \mathcal{Z}(y[n-1])) + \frac{1}{N}\left(\mathcal{Z}(x[n]) - \mathcal{Z}(x[n-N])\right) \\
&= z^{-1}Y(Z) + \frac{1}{N}\left(X(z) - z^{-N}X(z)\right)\right) \\
&= z^{-1}Y(Z) + \frac{1}{N}X(z)\left(1 - z^{-N}\right) \\
Y(z)\left(1 - z^{-1}\right) &= \frac{1}{N}X(z)\left(1 - z^{-N}\right) \\
Y(z) &= \frac{1}{N}X(z)\frac{1 - z^{-N}}{1 - z^{-1}}
\end{aligned}$$

And therefore:

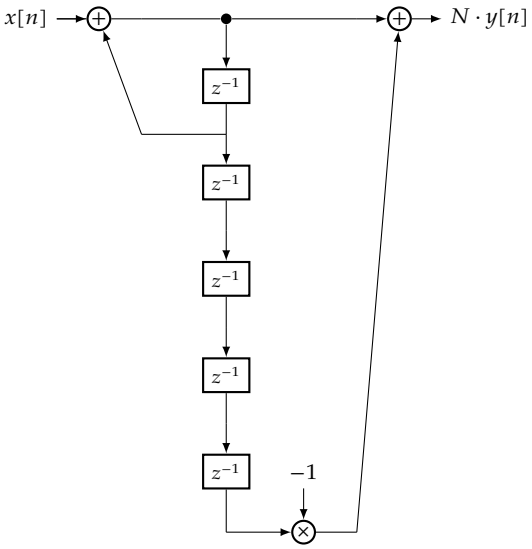
$$\begin{aligned}
H(z) &= \frac{Y(z)}{X(z)} \\
&= \frac{1}{N}\frac{1 - z^{-N}}{1 - z^{-1}}
\end{aligned}$$

The equivalence with the non-recursive implementation can be proven using long division.

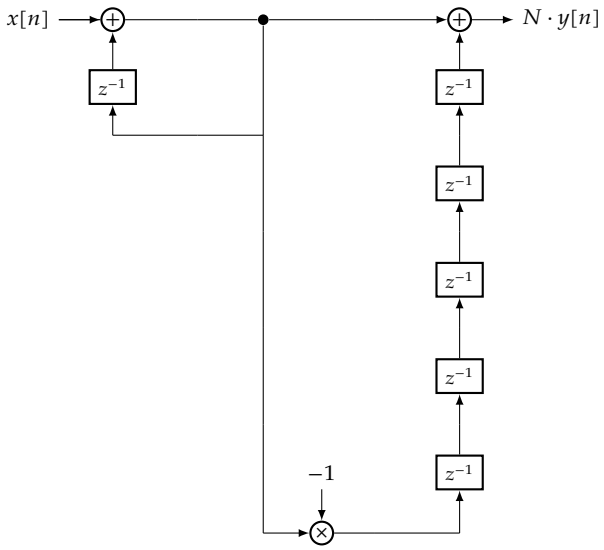
Solution 5.2.1.2-2: Let's draw a solution for $N = 5$. We omit the factor $1/N$ because it is only an overall scaling factor. The direct form I is easily found to be:



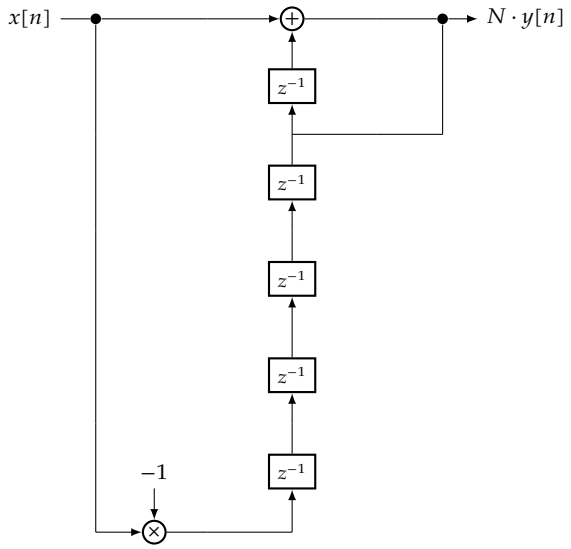
The direct form II is easily found to be:



The transposed form I is easily found to be:



The transposed form II is easily found to be:



Solution 5.2.1.3-1: To understand the functions used in the MATLAB/OCTAVE code below, read the final section in the chapter.

```
% H(z) = B(z) / A(z)
B = ones( 1, 5 );
A = zeros( 1, 5 );
A(1) = 1;
[ H, W ] = freqz (B, A, 1024, "whole" );
freqz_plot( W, H );
```

Solution 5.2.1.3-2: To understand the functions used in the MATLAB/OCTAVE code below, read the final section in the chapter.

```
N = 20;                     % just as an example
% H(z) = B(z) / A(z)
B = ones( 1, N );
A = zeros( 1, N );
A(1) = 1;
[ H, W ] = freqz (B, A, 1024, "whole" );
freqz_plot( W, H );
```

Chapter 6

FIR Filter Design

Solution 6.2.1-1: Let's start by incorporating all zeros in the transfer function:

$$\begin{aligned} H(z) &= (z - n_1)(z - n_2) \\ &= (z + 1.5)(z - 2.1) \\ &= z^2 - 0.6z - 3.15 \end{aligned}$$

This transfer function is not causal. Therefore, we define a new causal transfer function:

$$\begin{aligned} H_c(z) &= \frac{H(z)}{z^2} \\ &= \frac{z^2 - 0.6z - 3.1}{z^2} \\ &= 1 - 0.6z^{-1} - 3.1z^{-2} \end{aligned}$$

The spectrum can be generated in Octave using:

```
[h,w] = freqz( [ 1 -0.6 -3.1 ], [ 1 0 0 ], 256, "whole" );
freqz_plot( w, h );
```

Solution 6.2.1-2: Let's start by incorporating all zeros in the transfer function:

$$\begin{aligned} H(z) &= (z - n_1)(z - n_2) \\ &= (z - (-3 + \mathrm{j}2))(z - (-3 - \mathrm{j}2)) \\ &= z^2 + 6z + 13 \end{aligned}$$

This transfer function is not causal. Therefore, we define a new causal transfer function:

$$\begin{aligned} H_c(z) &= \frac{H(z)}{z^2} \\ &= \frac{z^2 + 6z + 13}{z^2} \\ &= 1 + 6z^{-1} + 13z^{-2} \end{aligned}$$

The spectrum can be generated in Octave using:

```
[h,w] = freqz( [ 1 6 13 ], [ 1 0 0 ], 256, "whole" );
freqz_plot( w, h );
```

Solution 6.2.1-3: Let's start by incorporating all zeros in the transfer function:

$$\begin{aligned} H(z) &= (z - n_1)(z - n_2)(z - n_3) \\ &= (z - 1.5)(z - (-3 + \mathrm{j}2))(z - (-3 - \mathrm{j}2)) \\ &= (z - 1.5)(z^2 + 6z + 13) \\ &= z^3 + 4.5z^2 + 4z - 19.5 \end{aligned}$$

This transfer function is not causal. Therefore, we define a new causal transfer function:

$$\begin{aligned} H_c(z) &= \frac{H(z)}{z^3} \\ &= \frac{z^3 + 4.5z^2 + 4z - 19.5}{z^3} \\ &= 1 + 4.5z^{-1} + 4z^{-2} - 19.5z^{-3} \end{aligned}$$

The spectrum can be generated in Octave using:

```
[h,w] = freqz( [ 1 4.5 4 -19.5 ], [ 1 0 0 0 ], 256, "whole" );
freqz_plot( w, h );
```

Solution 6.2.1-4: Let's start by incorporating all zeros in the transfer function:

$$\begin{aligned} H(z) &= (z - n_0)(z - n_1)(z - n_2)(z - n_3) \\ &= (z - 0.5)(z + 2)(z - (0.25 + \mathrm{j}3))(z - (0.25 - \mathrm{j}3)) \\ &= (z - 0.5)(z + 2)(z^2 - 0.5z + 9.0625 \\ &= (z - 0.5)(z^3 + 1.5z^2 + 8.0625z + 18.125) \\ &= z^4 + z^3 + 7.3125z^2 + 14.0938z - 9.0625 \end{aligned}$$

This transfer function is not causal. Therefore, we define a new causal transfer function:

$$\begin{aligned} H_c(z) &= \frac{H(z)}{z^4} \\ &= \frac{z^4 + z^3 + 7.3125z + 14.0938z - 9.0625}{z^4} \quad = 1 + z^{-1} + 7.3125z^{-2} + 14.0938z^{-3} - 9.0625z^{-4} \end{aligned}$$

The spectrum can be generated in Octave using:

```
[h,w] = freqz( [ 1 1 7.3125 14.0938 -9.0625 ], [ 1 0 0 0 0 ], 256, "whole" );
freqz_plot( w, h );
```

Solution 6.2.2-1: A feedforward comb filter has the following tranfer function:

$$H(z) = \frac{z^q + a}{z^q}$$

A filter with 6 dips is realized with $q = 6$. Using the comb ratio $R = 100$, we can calculate a to be:

$$a = \begin{cases} \frac{R-1}{R+1} & = \frac{99}{101} \\ \frac{R+1}{R-1} & = \frac{101}{99} \end{cases}$$

Both alternatives for a are valid.

The frequency response of the first alternative can be calculated with Octave:

```
N = 256;
[h,w] = freqz( [ 1 0 0 0 0 0 99/101 ], [ 1 0 0 0 0 0 0 ], N, "whole" );
freqz_plot( w, h );
```

The response of the second alternative can be calculated using:

```
[h,w] = freqz( [ 1 0 0 0 0 0 101/99 ], [ 1 0 0 0 0 0 0 ], N, "whole" );
freqz_plot( w, h );
```

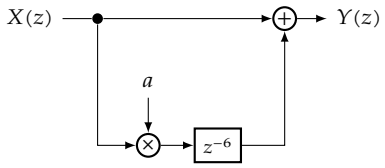
To be able to easily write a direct form, it is good practice to rewrite the transfer function a little bit:

$$\begin{aligned} H(z) &= \frac{z^q + a}{z^q} \\ &= 1 + az^{-q} \end{aligned}$$

As $H(z) = Y(z)/X(z)$ we can continue:

$$Y(z) = (1 + az^{-q})\, X(z)$$

From this expression, the direct form can readily be generated:



Moreover, transposing this form, yields the very same result!

Solution 6.2.2-2: A feedforward comb filter has the following tranfer function:

$$H(z) = \frac{z^q + a}{z^q}$$

A filter with 7 dips is realized with $q = 7$. Using the comb ratio $R = 0.05$, we can calculate a to be:

$$a = \begin{cases} \frac{R-1}{R+1} & = & \frac{-0.95}{1.05} \\ \frac{R+1}{R-1} & = & \frac{1.05}{-0.95} \end{cases}$$

Both alternatives for a are valid.

The frequency response of the first alternative can be calculated with Octave:

```
N = 256;
[h,w] = freqz( [ 1 0 0 0 0 0 -0.95/1.05 ], [ 1 0 0 0 0 0 0 ], N, "whole" );
freqz_plot( w, h );
```

The response of the second alternative can be calculated using:

```
[h,w] = freqz( [ 1 0 0 0 0 0 -1.05/0.95 ], [ 1 0 0 0 0 0 0 ], N, "whole" );
freqz_plot( w, h );
```

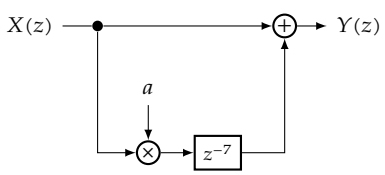
To be able to easily write a direct form, it is good practice to rewrite the transfer function a little bit:

$$\begin{aligned} H(z) &= \frac{z^q + a}{z^q} \\ &= 1 + az^{-q} \end{aligned}$$

As $H(z) = Y(z)/X(z)$ we can continue:

$$Y(z) = (1 + az^{-q})\, X(z)$$

From this expression, the direct form can readily be generated:



Moreover, transposing this form, yields the very same result!

Solution 6.2.2-3: In many cases the comb ratio is not given as such. This is such a case. To be able to calculate the comb ratio from the given design problem, it is paramount that we know how the comb ratio has been defined. It is the filter's magnitude response at $\omega = 0$ divided by the filter's magnitude resonse at the first peak/dip (whatever it may be) at $\omega = \omega_s/2q$.
Therefore:

$$R = \frac{2}{23}$$

Now, the design is most straightforward:

$$q = 42$$
$$a = \begin{cases} \frac{R+1}{R-1} = -1.1905 \\ \frac{R-1}{R+1} = -0.84 \end{cases}$$

Solution 6.3.2-1: Let's start by creating the 64 timepoints at a rate of 10 Hz:

```
N = 64;
t = linspace( 0, (N-1)*1/10, N );
```

Now, we can sample the impulse resonse:

```
h = cos( 2*pi*t ) .* exp( -5 * t );
```

You might consider plotting the result:

```
figure(1);
plot( t, h, 's' );
title( "Truncated␣Impulse␣Response" );
ylabel( "h_t[n]" );
xlabel( "n" );
```

Checking the frequency response by using the DtFT, can be done by using a zero-padded FFT (with an oversampling factor M):

```
M = 10;
w = linspace( 0, M*N-1, M*N );
figure(2);
subplot(2,1,1);
plot( w, abs( fft( h, M*N ) ) )
ylabel( "|H[k]|" );
xlabel( "M.k" );
title( "Frequency␣Response" );
subplot(2,1,2);
plot( w, angle( fft( h, M*N ) ) );
ylabel( "angle(H[k])" );
xlabel( "M.k" );
```

Solution 6.3.2-2: Let's start by creating the 32 timepoints at a rate of 1 Hz:

```
N = 32;
t = linspace( 0, (N-1), N );
```

Now, we can sample the impulse resonse:

```
h = t ./ (1+t.^2);
```

You might consider plotting the result:

```
figure(1);
plot( t, h, 's' );
title( "Truncated␣Impulse␣Response" );
ylabel( "h_t[n]" );
xlabel( "n" );
```

Checking the frequency response by using the DtFT, can be done by using a zero-padded FFT (with an oversampling factor M):

```
M = 10;
w = linspace( 0, M*N-1, M*N );
figure(2);
subplot(2,1,1);
plot( w, abs( fft( h, M*N ) ) );
ylabel( "|H[k]|" );
xlabel( "M.k" );
title( "Frequency␣Response" );
subplot(2,1,2);
plot( w, angle( fft( h, M*N ) ) );
ylabel( "angle(H[k])" );
xlabel( "M.k" );
```

The ripple that can be seen on magnitude and phase graph, can be reduced by applying a window function. To this end, we use the second half of a Hann window.

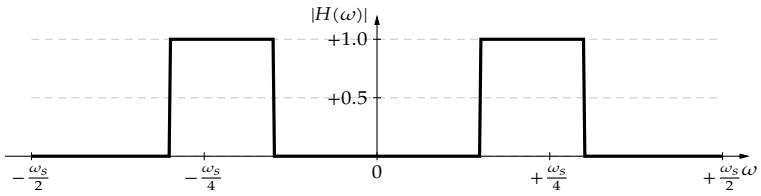
```
hwindow = hann( 2*N+1 )(N+1:2*N)';
hw = hann .* h;
```

Finally, we again, can check the frequency response using the DtFT:

```
figure(3);
subplot(2,1,1);
plot( w, abs( fft( hw, M*N ) ) );
ylabel( "|H_w[k]|" );
xlabel( "M.k" );
title( "Frequency␣Response" );
subplot(2,1,2);
plot( w, angle( fft( hw, M*N ) ) );
ylabel( "angle(H_w[k])" );
xlabel( "M.k" );
```

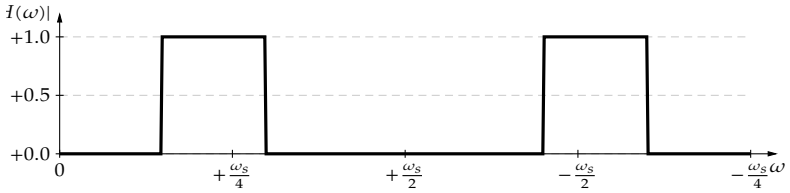
The ripple clearly is reduced.

Solution 6.4.2.2-1: The spectrum $|H(\omega)|$ looks like:

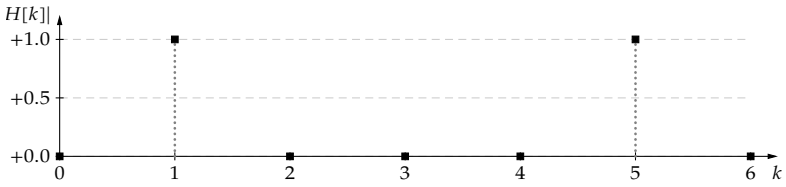


1. by hand for $N = 6$ (i.e. only using a calculator)

   1.1. Consider the spectrum from $\omega = 0$ to $\omega = \omega_s$.
   Graphically:



   1.2. Sample the (frequency-domain) magnitude spectrum
   Graphically:



1.3. Calculate the corresponding time-domain impulse response using the iFFT

		n					
		0	1	2	3	4	5
k	$X[k]$	W_6^{-kn}					
0	0	1.000	1.000	1.000	1.000	1.000	1.000
1	1	1.000	0.500+0.866j	-0.500+0.866j	-1.000+0.000j	-0.500-0.866j	0.500-0.866j
2	0	1.000	-0.500+0.866j	-0.500-0.866j	1.000-0.000j	-0.500+0.866j	-0.500-0.866j
3	0	1.000	-1.000+0.000j	1.000-0.000j	-1.000+0.000j	1.000-0.000j	-1.000+0.000j
4	0	1.000	-0.500-0.866j	-0.500+0.866j	1.000-0.000j	-0.500-0.866j	-0.500+0.866j
5	1	1.000	0.500-0.866j	-0.500-0.866j	-1.000+0.000j	-0.500+0.866j	0.500+0.866j
		$x[n]$					
		0.333	0.167	-0.167	-0.333+0.000j	-0.167+0.000j	0.167+0.000j

Resulting in:

$$h_1[n] = [\underbrace{0.333}_{n=0}, 0.167, -0.167, -0.333, -0.167, 0.167]$$

1.4. Move the samples at $n \geq N/2$ to $n - N$ to make the impulse response center-symmetric

$$h_2[n] = [-0.333, -0.167, 0.167, \underbrace{0.333}_{n=0}, 0.167, -0.167]$$

1.5. Shift the impulse response to make it causal

$$h_3[n] = [\underbrace{-0.333}_{n=0}, -0.167, 0.167, 0.333, 0.167, -0.167]$$

1.6. Check the result using the FFT

		k					
		0	1	2	3	4	5
n	$x[n]$	W_6^{kn}					
0	-0.33333333	1.000	1.000	1.000	1.000	1.000	1.000
1	-0.16666666667	1.000	0.500-0.866j	-0.500-0.866j	-1.000-0.000j	-0.500+0.866j	0.500+0.866j
2	0.16666666666667	1.000	-0.500-0.866j	-0.500+0.866j	1.000+0.000j	-0.500-0.866j	-0.500+0.866j
3	0.333333333	1.000	-1.000-0.000j	1.000+0.000j	-1.000-0.000j	1.000+0.000j	-1.000-0.000j
4	0.1666666666667	1.000	-0.500+0.866j	-0.500-0.866j	1.000+0.000j	-0.500+0.866j	-0.500-0.866j
5	-0.16666666666667	1.000	0.500+0.866j	-0.500+0.866j	-1.000-0.000j	-0.500-0.866j	0.500-0.866j
		$X[k]$					
		0.000	-1.000+0.000j	0.000+0.000j	0.000+0.000j	0.000-0.000j	-1.000-0.000j

This is what we expected.

1.7. Finally check the result using a zero-padded FFT
We won't perform this step, as the amount of calculations for any significant zero-padding exceeds what's reasonable.

1.8. Apply a window function to the resulting impulse response
We can calculate the required Blackmann window as

$$\begin{aligned}\text{Blackman}_6[n] &= 0.42 - 0.5\cos\left(\frac{2\pi n}{6}\right) + 0.08\cos\left(\frac{4\pi n}{6}\right) \qquad \text{if } 0 \leq n \leq 5 \\ &= [0, 0.13, 0.63, 1, 0.63, 0.13]\end{aligned}$$

Using this window, we can calculate the windowed impulse response to be:

$$h_w[n] = [0.00000, 0.02167, 0.10500, 0.33333, 0.10500, -0.02167]$$

1.9. Check the result using the FFT

		k					
		0	1	2	3	4	5
n	$x[n]$	W_6^{kn}					
0	0.00000	1.000	1.000	1.000	1.000	1.000	1.000
1	0.02167	1.000	0.500-0.866j	-0.500-0.866j	-1.000-0.000j	-0.500+0.866j	0.500+0.866j
2	0.10500	1.000	-0.500-0.866j	-0.500+0.866j	1.000+0.000j	-0.500-0.866j	-0.500+0.866j
3	0.33333	1.000	-1.000-0.000j	1.000+0.000j	-1.000-0.000j	1.000+0.000j	-1.000-0.000j
4	0.10500	1.000	-0.500+0.866j	-0.500-0.866j	1.000+0.000j	-0.500+0.866j	-0.500-0.866j
5	-0.02167	1.000	0.500+0.866j	-0.500+0.866j	-1.000-0.000j	-0.500-0.866j	0.500-0.866j
		$X[k]$					
		0.543	-0.438-0.038j	0.228-0.038j	-0.123+0.000j	0.228+0.038j	-0.438+0.038j

The result looks quite bad. This is due to the low number points we used to sample the spectrum. Therefore the main lobe of the Blackman window is relatively big compared to the frequency range.

1.10. Finally check the result using a zero-padded FFT
We won't perform this step, as the amount of calculations for any significant zero-padding exceeds what's reasonable.

2. using Octave for $N = 64$, following "the hard way"

2.1. Consider the spectrum from $\omega = 0$ to $\omega = \omega_s$.

2.2. Sample the (frequency-domain) magnitude spectrum
Let's start by generating the frequency response:

```
N = 64;
k = linspace( 0, N-1, N );
H = zeros( 1, N );
H( 1 + ceil(0.15 * N) : 1 + floor(0.3 * N) ) = 1;
H( N + 1 - floor(0.3 * N) : N + 1 - ceil(0.15 * N) ) = 1;
```

2.3. Calculate the corresponding time-domain impulse response using the iFFT

```
h = ifft( H );
```

2.4. Move the samples at $n \geq N/2$ to $n - N$ to make the impulse response center-symmetric

```
h = shift( h, floor(N/2) );
```

2.5. Shift the impulse response to make it causal This actually is a 'no-op'.

2.6. Check the result using the FFT

```
Hk = fft(h);
figure(1);
subplot( 2, 1, 1 );
plot( k, abs( Hk ), 's' );
title( "Filter␣Frequency␣Response" );
ylabel( "|H[k]|" );
xlabel( "k" );
subplot( 2, 1, 2 );
plot( k, angle( Hk ), 's' );
ylabel( "angle(H[k])" );
xlabel( "k" );
```

2.7. Finally check the result using a zero-padded FFT

```
M = 50;
w = linspace( 0, N*M-1, N*M );
HH = fft(h, M * N);
figure(2);
subplot( 2, 1, 1 );
plot( w, abs( HH ) );
title( "Filter␣Frequency␣Response␣(with␣M␣=␣50)" );
ylabel( "|H[k]|" );
xlabel( "M*k" );
subplot( 2, 1, 2 );
plot( w, angle( HH ) );
ylabel( "angle(H[k])" );
xlabel( "M*k" );
```

We can clearly see some severe Gibbs effect (a.k.a spectral leak, or Dirichlet-effect).

2.8. Apply a window function to the resulting impulse response
Beware the strange Octave/Matlab behavior w.r.t. windows!

```
wb = blackman(N+1)(1:N)';
hw = h .* wb;
```

2.9. Check the result using the FFT

```
H_w = fft(hw);
figure(3);
subplot( 2, 1, 1 );
plot( k, abs( H_w ), 's' );
title( "Filter␣Frequency␣Response␣(with␣Blackman␣window)" );
ylabel( "|H_w[k]|" );
xlabel( "k" );
subplot( 2, 1, 2 );
plot( k, angle( H_w ), 's' );
ylabel( "angle(H_w[k])" );
xlabel( "k" );
```

2.10. Finally check the result using a zero-padded FFT

```
HH_w = fft(hw, M * N);
figure(4);
subplot( 2, 1, 1 );
plot( w, abs( HH_w ) );
title( "Filter␣Frequency␣Response␣(with␣Blackmand␣window␣and␣M␣=␣50)" );
ylabel( "|H[k]|" );
xlabel( "M*k" );
subplot( 2, 1, 2 );
plot( w, angle( HH_w ) );
ylabel( "angle(H[k])" );
xlabel( "M*k" );
```

3. using Octave's `fir1` function for $N = 64$ You might want to start by reading the documentation for the Octave/Matlab function `fir1`:

```
help fir1;
```

And then we can design our filter using:

```
N = 64;
h = fir1( N-1, [ 0.3, 0.6 ], 'DC-0', 'boxcar' );
```

Or using a Blackman window:

```
N = 64;
hw = fir1( N-1, [ 0.3, 0.6 ], 'DC-0', 'blackman' );
```

The mechanics to check the resulting spectrum (or spectrum after zero-padding) is identical to what has been presented above.

Solution 6.4.2.2-2:

1. using Octave for $N = 256$, following "the hard way"

   1.1. Consider the spectrum from $\omega = 0$ to $\omega = \omega_s$.

   1.2. Sample the (frequency-domain) magnitude spectrum
   Let's start by generating the frequency response:

```
N = 256;
k = linspace( 0, N-1, N );
H = zeros( 1, N );
H( 1 : floor( 0.2 * N) ) = 0.5;
for i = ceil( 0.2 * N ) : floor( 0.4 * N )
  H(i) = -5/N*( i - 0.4*N);
end;
H(floor(N/2) + 2 : N) = H(ceil(N/2):-1:2);
```

   1.3. Calculate the corresponding time-domain impulse response using the iFFT

```
h = ifft( H );
```

   1.4. Move the samples at $n \geq \lfloor N/2 \rfloor$ to $n - N$ to make the impulse response center-symmetric

```
h = shift( h, floor(N/2) );
```

   1.5. Shift the impulse response to make it causal This actually is a 'no-op'.

   1.6. Check the result using the FFT

```
Hk = fft(h);
figure(1);
subplot( 2, 1, 1 );
plot( k, abs( Hk ), 's' );
title( "Filter␣Frequency␣Response" );
ylabel( "|H[k]|" );
xlabel( "k" );
subplot( 2, 1, 2 );
plot( k, angle( Hk ), 's' );
ylabel( "angle(H[k])" );
xlabel( "k" );
```

   1.7. Finally check the result using a zero-padded FFT

```
M = 20;
w = linspace( 0, N*M-1, N*M );
HH = fft(h, M * N);
figure(2);
subplot( 2, 1, 1 );
plot( w, abs( HH ) );
title( "Filter␣Frequency␣Response␣(with␣M␣=␣20)" );
ylabel( "|H[k]|" );
xlabel( "M*k" );
subplot( 2, 1, 2 );
plot( w, angle( HH ) );
ylabel( "angle(H[k])" );
xlabel( "M*k" );
```

   We can clearly see some severe Gibbs effect (a.k.a spectral leak, or Dirichlet-effect).

   1.8. Apply a window function to the resulting impulse response
   Beware the strange Octave/Matlab behavior w.r.t. windows!

```
wb = hamming(N+1)(1:N)';
hw = h .* wb;
```

   1.9. Check the result using the FFT

```
H_w = fft(hw);
figure(3);
subplot( 2, 1, 1 );
plot( k, abs( H_w ), 's' );
title( "Filter␣Frequency␣Response␣(with␣Blackman␣window)" );
```

```
ylabel( "|H_w[k]|" );
xlabel( "k" );
subplot( 2, 1, 2 );
plot( k, angle( H_w ), 's' );
ylabel( "angle(H_w[k])" );
xlabel( "k" );
```

1.10. Finally check the result using a zero-padded FFT

```
HH_w = fft(hw, M * N);
figure(4);
subplot( 2, 1, 1 );
plot( w, abs( HH_w ) );
title( "Filter␣Frequency␣Response␣(with␣Blackmand␣window␣and␣M␣=␣20)" );
ylabel( "|H[k]|" );
xlabel( "M*k" );
subplot( 2, 1, 2 );
plot( w, angle( HH_w ) );
ylabel( "angle(H[k])" );
xlabel( "M*k" );
```

2. using Octave's `fir2` function for $N = 256$ You might want to start by reading the documentation for the Octave/Matlab function `fir2`:

```
help fir2;
```

And then we can design our filter using:

```
N = 256;
h = fir2( N-1, [ 0, 0.4, 0.4, 0.8, 1 ], [0.5, 0.5, 1, 0, 0], 'boxcar' );
```

Or using a Hamming window:

```
N = 64;
h = fir2( N-1, [ 0, 0.4, 0.4, 0.8, 1 ], [0.5, 0.5, 1, 0, 0], 'hamming' );
```

The mechanics to check the resulting spectrum (or spectrum after zero-padding) is identical to what has been presented above.

Solution 6.5.2.3-1:

1. Using the grid method:
Let's start by defining $G(\tilde{\omega})$:

```
function y = gnormfreq( w )
  y = zeros(size(w));
  y+= ( abs(w) < 0.4*pi ) .* (1 - abs(w)/2/pi);
  y+= ( abs(w) >= 0.4*pi & abs(w) < 0.5*pi ) * 0.8;
  y+= ( abs(w) >= 0.5*pi & abs(w) < 0.6*pi ) .* ( 0.8 - 2/pi*(abs(w) - 0.5*pi) );
  y+= ( abs(w) >= 0.6*pi ) * 0.2;
end
```

Let's use this function to design an $2M + 1$-point filter to comply with an L-point grid.

```
M = 100;
N = 2*M+1;
L = 300;
wk = linspace( 0, pi, L )';
```

As the exercise did not mention values for these parameters, we chose them freely. In real life, you will determine these parameters, based on the quality of the filter you need and based on the processing power available.
And now, let's calculate the matrices $\boldsymbol{A}$ and $\vec{B}$:

```
A = cos( wk * (M:-1:0) );
B = gnormfreq( wk );
```

As one cannot check enough, let's check on the shape of $\vec{B}$ to see whether it corresponds to the original magnitude spectrum.

```
plot( wk, B );
```

Now, we can solve for $\vec{Q}$:

```
Q = A \ B;
```

Starting from Q, we can generate the impulse response (and make it causal at the same time):

```
h = zeros( N, 1 );
h(M+1) = Q(M+1);
for i = 1:M
  h(i) = h(N+1-i) = Q(i) / 2;
end
```

Let's take a look at the result:

```
plot( h, 's' );
ylabel( "h[n]" );
xlabel( "n" );
title( "Impulse␣response" );
```

And, let's check the resulting spectrum

```
H = shift( fft(h), floor(N/2) );
```

and plot it:

```
subplot(2,1,1);
plot( abs(H) );
ylabel( "|H(w)|" );
subplot(2,1,2);
plot( angle(H) );
ylabel( "angle(H(w))" );
xlabel( "w" );
title( "Resulting␣FFT␣spectrum" );
```

The Gibbs effect will only become visible after zero-padding. This effect may be smoothed out using a window function. E.g., a Kaiser window.:

```
hw = h .* kaiser( N + 1 )(1:N);
```

2. Using the `firls` method:

```
M = 200;
h = firls(M, [ 0 0.4 0.4 0.5 0.5 0.6 0.6 1.0 ],
             [ 1 0.8 0.8 0.8 0.8 0.6 0.2 0.2 ] );
```

Solution 6.5.2.3-2:

1. Using the grid method:
   Let's start by defining $G(\tilde{\omega})$:

```
function y = gnormfreq( w )
  y = zeros(size(w));
  y+= ( abs(w) < 0.2*pi ) * 1;
  y+= ( abs(w) >= 0.2*pi & abs(w) < 0.4*pi ) * 0.25;
  y+= ( abs(w) >= 0.4*pi & abs(w) < 0.6*pi ) * 1;
  y+= ( abs(w) >= 0.8*pi ) * 1;
end
```

Let's use this function to design an $2M + 1$-point filter to comply with an L-point grid.

```
M = 25;
N = 2*M+1;
L = 100;
wk = linspace( 0, pi, L )';
```

As the exercise did not mention values for these parameters, we chose them freely. In real life, you will determine these parameters, based on the quality of the filter you need and based on the processing power available.

And now, let's calculate the matrices $\boldsymbol{A}$ and $\vec{B}$:

```
A = cos( wk * (M:-1:0) );
B = gnormfreq( wk );
```

As one cannot check enough, let's check on the shape of $\vec{B}$ to see whether it corresponds to the original magnitude spectrum.

```
plot( wk, B, 's' );
```

Now, we can solve for $\vec{Q}$:

```
Q = A \ B;
```

Starting from Q, we can generate the impulse response (and make it causal at the same time):

```
h = zeros( N, 1 );
h(M+1) = Q(M+1);
for i = 1:M
  h(i) = h(N+1-i) = Q(i) / 2;
end
```

Let's take a look at the result:

```
plot( h, 's' );
ylabel( "h[n]" );
xlabel( "n" );
title( "Impulse␣response" );
```

And, let's check the resulting spectrum

```
H = shift( fft(h), floor(N/2) );
```

and plot it:

```
subplot(2,1,1);
plot( abs(H), 's' );
ylabel( "|H(w)|" );
subplot(2,1,2);
plot( angle(H), 's' );
ylabel( "angle(H(w))" );
xlabel( "w" );
title( "Resulting␣FFT␣spectrum" );
```

The result does not look too good. Increasing M (if you can afford it in terms of memory and processing power), is a good idea. In addition, you might consider adding a weight function to avoid the ringing near the band edges.

The true Gibbs effect will only become visible after zero-padding. This effect may be smoothed out using a window function. E.g., a Kaiser window.:

```
hw = h .* kaiser( N + 1 )(1:N);
```

2. Using the `firls` method:

```
M = 50;
h = firls(M, [ 0 0.2 0.2 0.4 0.4 0.6 0.6 0.8 0.8 1.0 ],
             [ 1 1 0.25 0.25 1   1   0   0   1   1   ] );
```

Solution 6.5.3.2-1: Let's start by composing a graphical jig, to easily check our results. We'll do this by composing a function taking as argument the impulse response (h).

```
function plotgraphonspecs( h )
  hh     = zeros( 1, 10*length(h) );
  hh(1:length(h)) = h;
  N      = length(hh);
  Hmagdb = 20*log10(abs(fft(hh)));
  w      = 1:length(hh);
  wnyq   = N/2;             % nyquist frequency
  hold off;
  plot( w, Hmagdb );
  hold on;
  wpl = round(0.2*wnyq)  + 1;  % + 1 due to vector indices 1:N
  wsl = round(0.3*wnyq)  + 1;
  wsh = round(0.56*wnyq) + 1;
  wph = round(0.74*wnyq) + 1;
  passrange = [ 1:wpl, wph:N+2-wph, N+2-wpl:N ];
  stoprange = [ wsl:wsh, N+2-wsh:N+2-wsl];
  xlabel( "k␣␣␣(1:N)" );
  ylabel( "|H[k]|␣␣␣(dB)" );
  title( "Filter␣characteristic␣compared␣to␣the␣specifications" );
  plot( passrange,  0.5 * ones(size(passrange)), 'o', "markersize", 2 );
  plot( passrange, -0.5 * ones(size(passrange)), 'o', "markersize", 2 );
  plot( stoprange, -20 * ones(size(stoprange)), 'o', "markersize", 2  );
  if ( max( Hmagdb(passrange) < -0.5 ) == 1 )
    printf( "Spec␣violation␣in␣passband␣on␣lower␣bound\n" );
  end
  if ( max( Hmagdb(passrange) > 0.5 ) == 1 )
    printf( "Spec␣violation␣in␣passband␣on␣upper␣bound\n" );
  end
  if ( max( Hmagdb(stoprange) > -20 ) == 1 )
    printf( "Spec␣violation␣in␣stopband␣on␣upper␣bound\n" );
  end
end
```

We added a bit of zero-padding (to mimick the DtFT) before plotting the specs in order to more accurately check the band boundaries. You might think that we applied zeropadding to increase the frequency resolution in order not to miss any Gibbs effect. However, the Remez-exchange algorithm fixes the extremals, and therefore Gibbs effect is unlikely to be a problem.
We can test this jig function by applying it on a random impulse response. E.g.,

```
plotgraphonspecs( rand( 1, 256 ) );
```

Next, let's start designing our filter.

```
f = [ 0.000 0.200 0.300 0.560 0.740 1.000 ];
a = [ 1.000 1.000 0.000 0.000 1.000 1.000 ];
w = [ 1.000       1.000       1.000       ];
b = remez( 32, f, a, w );
```

Then, we can test the result:

```
plotgraphonspecs( b );
```

This solution is completely within spec, so we may try to reduce the number of filter coefficients. This can be done easily:

```
b = remez( 31, f, a, w );
plotgraphonspecs( b );
```

Oops! What happened? The filter now is totally out of spec? Well, you bumped into one of the peculiarities of digital filter design. A bit of explanation is needed. Calling the `remez` function to generate a filter with N taps, generates an impulse response of length $N + 1$. This means that in the previous case, you obtain an impulse response length of 32.
In general, designing high-pass filters with an even impulse response length is most difficult. Why?

An even impulse response length implies that the frequency sampling of the spectrum (the FFT of the impulse response) exhibits a point exactly at the Nyquist frequency. As the spectrum of a real function is Hermitic, we must have:

$$H(\omega_N) = \overline{H(-\omega_N)}$$

But because the spectrum of a sampled signal is also periodic, we als require:

$$H(\omega_N) = H(-\omega_N)$$

The only solution to this set of equations is that $H(\omega_N) = H(-\omega_N) \in \mathbb{R}$. Moreover, our specs require $|H(\omega_N)| = 1$. This requires the phase of our filter to be an exact multiple of 2π at Nyquist frequency. This is an extreme harsh condition that the Remez-algorithm cannot overcome. Note that this is not a problem of the algorithm, but a problem introduced by taking a even length impulse response. Therefore, we stick to odd-length impulse responses, and continue:

```
b = remez( 30, f, a, w );
plotgraphonspecs( b );
```

OK. The even-length problems are gone. This filter is again within spec. One might not be so happy about the bump in the second transition band. It depends on the application whether you can tolerate this or not. We may continue lowering the filter length:

```
b = remez( 28, f, a, w );
plotgraphonspecs( b );
```

Still within spec. Even further:

```
b = remez( 26, f, a, w );
plotgraphonspecs( b );
```

Strangely, the nasty transition band effect is gone. Welcome to Remez' world. We continue lowering the filter length.

```
b = remez( 24, f, a, w );
plotgraphonspecs( b );
b = remez( 22, f, a, w );
plotgraphonspecs( b );
b = remez( 20, f, a, w );
plotgraphonspecs( b );
```

And then we again have problems in the passband. Now, we can try to fiddle with the weights.

```
w = [ 2.000       1.000       2.000       ];
b = remez( 20, f, a, w );
plotgraphonspecs( b );
```

And the problems are again solved. It is easy to see that this kind of optimization is not easily guided by plan, and can take quite some time. A trade-off needs be made between engineering time required to find the optimal solution and the opportunity cost that bares on the final product. Though it is possible to continue, we'll stop here. Feel tempted to do better than this last solution.

Solution 6.5.3.2-2: Let's start by composing a graphical jig, to easily check our results. We'll do this by composing a function taking as argument the impulse response (h).

```
function plotgraphonspecs( h )
  hh     = zeros( 1, 10*length(h) );
  hh(1:length(h)) = h;
  N      = length(hh);
  Hmagdb = 20*log10(abs(fft(hh)));
  w      = 1:length(hh);
  wnyq   = N/2;             % nyquist frequency
  hold off;
```

```
  plot( w, Hmagdb );
  hold on;
  wsl = round(0.2*wnyq) + 1;  % + 1 due to vector indices 1:N
  wpl = round(0.3*wnyq) + 1;
  wph = round(0.5*wnyq) + 1;
  wsh = round(0.7*wnyq) + 1;
  passrange = [ wpl:wph, N+2-wph:N+2-wpl];
  stoprange = [ 1:wsl, wsh:N+2-wsh, N+2-wsl:N ];
  xlabel( "k␣␣␣(1:N)" );
  ylabel( "|H[k]|␣␣␣(dB)" );
  title( "Filter␣characteristic␣compared␣to␣the␣specifications" );
  plot( passrange,  1.0 * ones(size(passrange)), 'o', "markersize", 2 );
  plot( passrange, -1.0 * ones(size(passrange)), 'o', "markersize", 2 );
  plot( stoprange, -40 * ones(size(stoprange)), 'o', "markersize", 2  );
  if ( max( Hmagdb(passrange) < -1 ) == 1 )
    printf( "Spec␣violation␣in␣passband␣on␣lower␣bound\n" );
  end
  if ( max( Hmagdb(passrange) > 1 ) == 1 )
    printf( "Spec␣violation␣in␣passband␣on␣upper␣bound\n" );
  end
  if ( max( Hmagdb(stoprange) > -40 ) == 1 )
    printf( "Spec␣violation␣in␣stopband␣on␣upper␣bound\n" );
  end
end
```

We added a bit of zero-padding (to mimick the DtFT) before plotting the specs in order to more accurately check the band boundaries. You might think that we applied zeropadding to increase the frequency resolution in order not to miss any Gibbs effect. However, the Remez-exchange algorithm fixes the extremals, and therefore Gibbs effect is unlikely to be a problem.
We can test this jig function by applying it on a random impulse response. E.g.,

```
plotgraphonspecs( rand( 1, 256 ) );
```

Next, let's start designing our filter.

```
f = [ 0.000 0.200 0.300 0.500 0.700 1.000 ];
a = [ 0.000 0.000 1.000 1.000 0.000 0.000 ];
w = [ 1.000       1.000       1.000       ];
b = remez( 32, f, a, w );
```

Then, we can test the result:

```
plotgraphonspecs( b );
```

This solution is out of spec in the stop band, but there is clearly some margin in the pass band. We can try to fiddle with the weights. This can be done easily:

```
w = [ 2.000       1.000       2.000       ];
b = remez( 32, f, a, w );
plotgraphonspecs( b );
```

Still not within spec in the stop-band. Some more:

```
w = [ 4.000       1.000       4.000       ];
b = remez( 32, f, a, w );
plotgraphonspecs( b );
```

Still not within spec. Some more:

```
w = [ 6.000       1.000       6.000       ];
b = remez( 32, f, a, w );
plotgraphonspecs( b );
```

OK, now we are within spec. One might not be so happy about the bump in the second transition band. It depends on the application whether you can tolerate this or not. One way to avoid it is to reduce the

transition bands (see the table with hints in the course notes). However, an alternative is to lower the filter length by 1 or 2. Let's try:

```
b = remez( 31, f, a, w );
plotgraphonspecs( b );
```

Still within spec. But the bump is still there. Even further:

```
b = remez( 30, f, a, w );
plotgraphonspecs( b );
```

The bump is now gone, but there is again a violation in the stop-band. A good step would be to increase the length of the first stop-band a bit:

```
f = [ 0.000 0.210 0.300 0.500 0.700 1.000 ];
b = remez( 30, f, a, w );
plotgraphonspecs( b );
```

And then increase the stop-band weight a bit:

```
w = [ 10.000       1.000       10.000       ];
b = remez( 30, f, a, w );
plotgraphonspecs( b );
```

To no avail. A next possible step is to increase the upper pass-band edge a bit.

```
f = [ 0.000 0.210 0.300 0.510 0.700 1.000 ];
b = remez( 30, f, a, w );
plotgraphonspecs( b );
```

And the filter is again within spec. It is easy to see that this kind of optimization is not easily guided by plan, and can take quite some time. A trade-off needs be made between engineering time required to find the optimal solution and the opportunity cost that bares on the final product. Though it is possible to continue, we'll stop here. Feel tempted to do better than this last solution.

Chapter 7

IIR Filter Design

Solution 7.2.1-1: Let's start by incorporating all zeros and poles in the transfer function:

$$\begin{aligned} H(z) &= \frac{z - n_1}{z - p_1} \\ &= \frac{z + 1.5}{z - 0.1} \end{aligned}$$

The spectrum can be generated in Octave using:

```
[h,w] = freqz( [ 1 1.5 ], [ 1 -2.1 ], 256, "whole" );
freqz_plot( w, h );
```

Solution 7.2.1-2: The design of this filter is a bad idea, as it exhibits a pole outside the unit circle and hence will not be stable.

Solution 7.2.1-3: Let's start by incorporating all zeros and poles in the transfer function:

$$\begin{aligned} H(z) &= \frac{(z - n_1)(z - n_2)}{(z - p_0)(z - p_1)(z - p_2)} \\ &= \frac{(z + 3 - \mathrm{j}2)(z + 3 + \mathrm{j}2)}{(z + 0.3)(z - 0.5 - \mathrm{j}0.8)(z - 0.5 + \mathrm{j}0.8)} \\ &= \frac{z^2 + 6z + 13}{(z + 0.3)(z^2 - z + 0.89)} \\ &= \frac{z^2 + 6z + 13}{z^3 - 0.7z^2 + 0.59z + 0.267} \end{aligned}$$

This transfer function is causal as the degree of the denominator is higher than the degree of the numerator. The spectrum can be generated in Octave using:

```
[h,w] = freqz( [ 1 6 13 ], [ 1 -0.7 0.59 0.267 ], 256, "whole" );
freqz_plot( w, h );
```

Solution 7.2.3-1:

1. Design
   If we need a feedback comb filter with 5 peaks, this means the delay in the feedback loop needs to be five time slots, i.e. $q = 5$. In Octave:

   ```
   q = 5;
   ```

In addition, we can use the design equation derived for feedback comb filters:

$$a = \frac{R-1}{R+1}$$

In Octave:

```
R = 200;
a = ( R - 1 ) / ( R + 1 );
```

Given $R = 200$, this leads to $a = 0.99005$. This finishes our design.

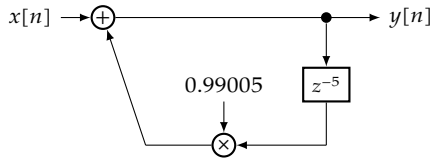
2. Verify the frequency response
Let's check the frequency response in Octave. This is easily done, by considering the transfer function of the filter we designed in the Z-domain:

$$H(z) = \frac{z^q}{z^q - a}$$

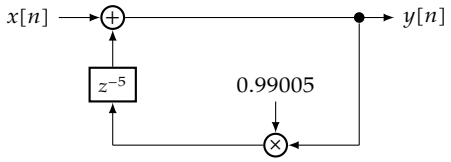
Using the appropriate functions to plot frequency responses in Octave, it's easy to plot the resulting frequency response.

```
B = zeros( 1, q+1 );
B(1) = 1;
A = zeros( 1, q+1 );
A(1) = 1;
A(q+1) = -a;
[ H, W ]  = freqz( B, A, 2048, "whole" );
freqz_plot( W, H );
```

3. Compose a direct form I and a tranposed form I implementation of the filter.
The direct form I implementation:



The transposed form I implementation:



As you will have noticed, the implementations are identical. This is a peculiarity of this type of comb filter.

Solution 7.2.3-2:

1. Design
If we need a feedback comb filter with 8 peaks, this means the delay in the feedback loop needs to be five time slots, i.e. $q = 8$. In Octave:

```
q = 8;
```

In addition, we can use the design equation derived for feedback comb filters:

$$a = \frac{R-1}{R+1}$$

In Octave:

```
R = 0.02;
a = ( R - 1 ) / ( R + 1 );
```

Given $R = 200$, this leads to $a = -0.96078$. This finishes our design.

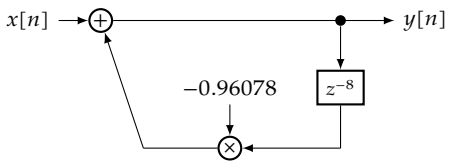
2. Verify the frequency response
   Let's check the frequency response in Octave. This is easily done, by considering the transfer function of the filter we designed in the Z-domain:

$$H(z) = \frac{z^q}{z^q - a}$$

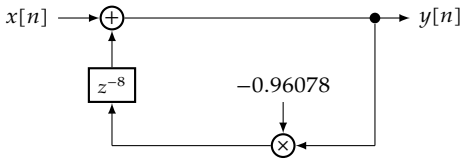
   Using the appropriate functions to plot frequency responses in Octave, it's easy to plot the resulting frequency response.

```
B = zeros( 1, q+1 );
B(1) = 1;
A = zeros( 1, q+1 );
A(1) = 1;
A(q+1) = -a;
[ H, W ]  = freqz( B, A, 2048, "whole" );
freqz_plot( W, H );
```

3. Compose a direct form II and a tranposed form II implementation of the filter.
   The direct form II implementation:



   The transposed form II implementation:



   As you will have noticed, the implementations are identical. Besides, you will notice that there is no difference between the I and II-type implementations. This is a peculiarity of this type of comb filter. It has only one basic form variant.

Solution 7.2.4.4-1: Using Octave, this design is most simple.

```
N = 8;   % order of the filter
         % (butter multiplies it by 2 for a bandstop filter)
[ B, A ] = butter( N, [0.12, 0.3], 'stop' );
```

For your reference, we list the coefficients in the table below:

	Coefficient of z^i	
i	numerator	denominator
16	2.2879e-01	1.0000e+00
15	-3.0121e+00	-1.0783e+01
14	1.9179e+01	5.6225e+01
13	-7.8185e+01	-1.8788e+02
12	2.2796e+02	4.4988e+02
11	-5.0339e+02	-8.1781e+02
10	8.7012e+02	1.1668e+03
9	-1.2002e+03	-1.3321e+03
8	1.3345e+03	1.2296e+03
7	-1.2002e+03	-9.2061e+02
6	8.7012e+02	5.5727e+02
5	-5.0339e+02	-2.6993e+02
4	2.2796e+02	1.0262e+02
3	-7.8185e+01	-2.9622e+01
2	1.9179e+01	6.1299e+00
1	-3.0121e+00	-8.1381e-01
0	2.2879e-01	5.2345e-02

We can easily check the frequency response of this filter:

```
[H,W] = freqz( B, A, 1024, "whole" );
freqz_plot( W, H );
```

The order of this filter is quite high. Therefore, we might end-up in the (non)stability danger zone due to coefficient rounding. Therefore, it's better to calculate poles and zeros, to be able to compose biquad sections.

```
[Z, P, g] = butter( N, [0.12, 0.3], 'stop' );
```

The result is:

i	pole	zero
0	0.89915 - j 0.35725	0.82283 + j 0.56829
1	0.83569 - j 0.34243	0.82283 - j 0.56829
2	0.76450 - j 0.33866	0.82283 + j 0.56829
3	0.68140 - j 0.35597	0.82283 - j 0.56829
4	0.68140 + j 0.35597	0.82283 + j 0.56829
5	0.76450 + j 0.33866	0.82283 - j 0.56829
6	0.83569 + j 0.34243	0.82283 + j 0.56829
7	0.89915 + j 0.35725	0.82283 - j 0.56829
8	0.55266 + j 0.74875	0.82283 + j 0.56829
9	0.52250 + j 0.62492	0.82283 - j 0.56829
10	0.53893 + j 0.50812	0.82283 + j 0.56829
11	0.59683 + j 0.41235	0.82283 - j 0.56829
12	0.59683 - j 0.41235	0.82283 + j 0.56829
13	0.53893 - j 0.50812	0.82283 - j 0.56829
14	0.52250 - j 0.62492	0.82283 + j 0.56829
15	0.55266 - j 0.74875	0.82283 - j 0.56829

This function returns the zeros and poles as vectors `Z` and `P` and the gain `g` as a scalar. Combining conjugate poles and zeros allows you to create biquad sections.

Solution 7.2.4.4-2: We're going to use OCTAVE to design this filter using the `cheby2` function. Let's start by defining a function that allows us checking the specs easily.

```
function plotgraphonspecs( B, A, N )
  [ H, W ] = freqz( B, A, N );
  H = H';
  W = W';
  Hmagdb = 20*log10(abs(H));
```

```
  ws = 0.29*pi;
  wp = 0.3*pi;
  hold off;
  plot( W, Hmagdb );
  hold on;
  stoprange = W < ws;
  passrange = W > wp;
  xlabel( "wnorm␣␣␣(0-ws)" );
  ylabel( "|H[k]|␣␣␣(dB)" );
  title( "Filter␣characteristic␣(green)␣compared␣to␣the␣specifications␣(blue)" );
  plot( W .* passrange,  zeros(size(passrange)), 'o', "markersize", 2 );
  plot( W .* passrange,  -3 * ones(size(passrange)), 'o', "markersize", 2 );
  plot( W .* stoprange, -46 * ones(size(stoprange)), 'o', "markersize", 2  );
  plot( W, Hmagdb, "2" );
  if ( max( Hmagdb(passrange) > sqrt(eps) ) == 1 )
    printf( "Spec␣violation␣in␣passband␣on␣upper␣bound:␣%f␣dB\n",
            max( Hmagdb( passrange ) ) );
  end
  if ( max( Hmagdb(passrange) < -3 - sqrt(eps) ) == 1 )
    printf( "Spec␣violation␣in␣passband␣on␣lower␣bound:␣%f␣dB\n",
            min( Hmagdb( passrange ) ) );
  end
  if ( max( Hmagdb(stoprange) > -46 + sqrt(eps) ) == 1 )
    printf( "Spec␣violation␣in␣stopband␣on␣upper␣bound:␣%f␣dB\n",
            max( Hmagdb( stoprange ) ) );
  end
end
```

Next, let's try to generate a filter of order 4:

```
[ B, A ] = cheby2( 4, 46, 0.29, 'high' );
plotgraphonspecs( B, A, 1024 );
```

You'll see that the roll-off slope is not high enough to comply with the pass-band specs. We'll increase the order of the filter, to improve the roll-off.

```
[ B, A ] = cheby2( 8, 46, 0.29, 'high' );
plotgraphonspecs( B, A, 1024 );
```

And again...

```
[ B, A ] = cheby2( 12, 46, 0.29, 'high' );
plotgraphonspecs( B, A, 1024 );
```

And again...

```
[ B, A ] = cheby2( 16, 46, 0.29, 'high' );
plotgraphonspecs( B, A, 1024 );
```

And again...

```
[ B, A ] = cheby2( 20, 46, 0.29, 'high' );
plotgraphonspecs( B, A, 1024 );
```

```
[ B, A ] = cheby2( 24, 46, 0.29, 'high' );
plotgraphonspecs( B, A, 1024 );
```

Now, let's decrease the order.

```
[ B, A ] = cheby2( 23, 46, 0.29, 'high' );
plotgraphonspecs( B, A, 1024 );
```

Alas, this does not work. Let's return to using order $N = 24$.

```
[ B, A ] = cheby2( 24, 46, 0.29, 'high' );
plotgraphonspecs( B, A, 1024 );
```

Now, we've only got a stopband violation. This is due to the fact that the `cheby2` Octave function does not perform as expected. The order of the filter (and the bad condition with respect to round-off errors on the location of poles and zeros) may be kicking in. Let's create some slack in the stopband, by changing the specs to the design function `cheby2`.

```
[ B, A ] = cheby2( 24, 47, 0.29, 'high' );
plotgraphonspecs( B, A, 1024 );
```

OK. Now the filter is within spec. We may continue, tweaking, but that probably is a waste of time.

Let's - because it is requested - increase the filter's order to $N = 50$:

```
[ B, A ] = cheby2( 50, 47, 0.29, 'high' );
plotgraphonspecs( B, A, 1024 );
```

Oops, we clearly hit an order for which the `cheby2` function fails. This - once again - teaches us to always check the result of an design algorithm before using it.

Solution 7.2.4.4-3: Octave's `ellip` function allows you to design Cauer filters. Let's calculate the passband corner frequencies relative to $omega_s/2$ to prepare the data for the `ellip` function.

```
Ws = 50e3;
Wpl = 5e3 / Ws * 2;
Wph = 12.5e3 / Ws * 2;
Rp = 0.5;            % passband ripple
Rs = 80;             % stopband ripple
```

And then, we can design the filter for various N. Let's start with defining a convenience function:

```
function [ W, H ] = designCauerBPF( N, Rp, Rs, Wpl, Wph, nofPoints = 1024 )
  [ B, A ] = ellip( N, Rp, Rs, [ Wpl, Wph ] );
  [ H, W ] = freqz( B, A, nofPoints );
  plot( W, 20*log10(H) );
end
```

Now, one option could be to manually determine these pass-band and stop-band corner frequencies. However, let's be a bit more ambitious and write a function that does the search for us.
The corner frequencies of the pass band can be defined as the extreme frequencies f for which holds $20\log(|H(f)|) \geq -3\,\text{dB}$. The corner frequencies of the stop band can be defined as the frequencies f closest to the pass-band frequencies for which holds for which holds: $20\log(|H(f)|) \leq -R_s$. This requirement is easily translated in Octave/Matlab, using the `find` function. Read the help on find to understand the function below.

```
function [ Wsl, Wpl, Wph, Wsh ] = findBPFCornerFrequencies( W, H, Rp, Rs )

  Hdb = 20*log10(abs(H)); % convert to magnitude on decibel scale

  % calculate pass band corners
  pBandIndices = find( Hdb > -Rp );
  Wpl = W( pBandIndices( 1 ) );
  Wph = W( pBandIndices( length(pBandIndices) ) );

  % calculate stop band corners
  sBandIndices = find( Hdb <= -Rs );
  losBandIndices = find( W(sBandIndices) <= Wpl );
  upsBandIndices = find( W(sBandIndices) >= Wph );
  Wsl = W( sBandIndices( losBandIndices( length(losBandIndices) ) ) );
  Wsh = W( sBandIndices( upsBandIndices( 1 ) ) );

end
```

Now, we can use the functions we defined to design the filter and determine the corner frequencies for all values of N.

```
for N = [ 2, 4, 6, 8 ]
  printf( "----------------------------------------\n");
  printf( "N␣␣␣␣␣␣␣␣Wsl␣␣␣␣␣␣␣␣Wpl␣␣␣␣␣␣␣␣Wph␣␣␣␣␣␣␣␣Wsh\n");
  printf( "----------------------------------------\n");
  [ W, H ] = designCauerBPF( N, Rp, Rs, Wpl, Wph );
  [ calcWsl, calcWpl, calcWph, calcWsh ] = ...
    findBPFCornerFrequencies( W, H, Rp, Rs );
  printf( "%d␣␣%8g␣␣%8g␣␣%8g␣␣%8g\n", N, calcWsl / pi * Ws / 2, ...
                                        calcWpl / pi * Ws / 2, ...
                                        calcWph / pi * Ws / 2, ...
                                        calcWsh / pi * Ws / 2 );
  printf( "----------------------------------------\n\n");
  pause();
end
```

This leads to the following result:

N	f_{SL}	f_{PL}	f_{PU}	f_{SU}
	[Hz]			
2	73	5005	12476	24731
4	1587	5005	12476	20239
6	3393	5005	12500	15625
8	4321	5005	12476	13721

Solution 7.3.3-1: Let's start by applying partial fraction expansion:

```
[R, P, K, E] = residue( [ 1 -1 -6], [1 3 5 -1] );
```

The values of R, P, K and E allow writing:

$$H(s) = \frac{0.99786 + 0.18588i}{s - (-1.58975 + \mathrm{j}1.74454)} + \frac{0.99786 - 0.18588i}{s - (-1.58975 - \mathrm{j}1.74454)} - \frac{0.99572}{s + 0.17951}$$

Applying the inverse Laplace transform results in:

$$\begin{aligned} h(t) =& u(t) \left(0.99572\,\mathrm{e}^{-0.17951t}\right. \\ &\left. + (0.99786 + 0.18588i)\,\mathrm{e}^{(-1.58975+\mathrm{j}1.74454)t} + (0.99786 - 0.18588i)\,\mathrm{e}^{(-1.58975-\mathrm{j}1.74454)t}\right) \end{aligned}$$

Sampling this continuous-time impulse response with $T_s = 0.1\,\mathrm{s}$ results in:

$$\begin{aligned} h[n] =& u[n] \left(0.99572\,\mathrm{e}^{-0.017951n}\right. \\ &\left. + (0.99786 + \mathrm{j}0.18588)\,\mathrm{e}^{(-0.158975+\mathrm{j}0.174454)n} + (0.99786 - \mathrm{j}0.18588)\,\mathrm{e}^{(-0.158975-\mathrm{j}0.174454)n}\right) \end{aligned}$$

Applying the Z-transform yields:

$$H(z) = \frac{0.99572z}{z - \mathrm{e}^{-0.017951}} + \frac{(0.99786 + \mathrm{j}0.18588)z}{z - \mathrm{e}^{(-0.158975+\mathrm{j}0.174454)}} + \frac{(0.99786 - \mathrm{j}0.18588)z}{z - \mathrm{e}^{(-0.158975-\mathrm{j}0.174454)}}$$

In view of the low order of the filter, we don't stick to biquad sections, but we transform this into a single transfer function. This is done by multiplying out the above equation. Let's engage Octave to do the work for us. We'll label the terms in the equation above to allow easy translation of the equation into the Octave code.

$$H(z) = \frac{\overbrace{0.99572z}^{A_N}}{\underbrace{z - \mathrm{e}^{-0.017951}}_{A_D}} + \frac{\overbrace{(0.99786 + \mathrm{j}0.18588)z}^{B_N}}{\underbrace{z - \mathrm{e}^{(-0.158975+\mathrm{j}0.174454)}}_{B_D}} + \frac{\overbrace{(0.99786 - \mathrm{j}0.18588)z}^{C_N}}{\underbrace{z - \mathrm{e}^{(-0.158975-\mathrm{j}0.174454)}}_{C_D}}$$

This allows rewriting $H(z)$ into:

$$H(z) = \frac{A_N}{A_D} + \frac{B_N}{B_D} + \frac{C_N}{C_D}$$

And further:

$$H(z) = \frac{\overbrace{A_N B_D C_D}^{R} + \overbrace{A_D B_N C_D}^{S} + \overbrace{A_D B_D C_N}^{T}}{\underbrace{A_D B_D C_D}_{Q}} = \frac{N(z)}{D(z)}$$

Now, the Octave code:

```
AN = [ 0.99572 0 ];
AD = [ 1 -e^(-0.017951) ];
BN = [ (0.99786 + 0.18588j) 0 ];
BD = [ 1 -e^(-0.17951 + 0.174454j ) ];
CN = [ (0.99786 - 0.18588j) 0 ];
CD = [ 1 -e^(-0.17951 - 0.174454j ) ];

R = conv( conv( AN, BD ), CD );
S = conv( conv( AD, BN ), CD );
T = conv( conv( AD, BD ), CN );
Q = conv( conv( AD, BD ), CD );

Nz = R + S + T
==> Nz = 2.99144  -5.29555   2.36158   0.00000
Dz = Q
==> Dz = 1.00000  -2.62820   2.31507  -0.68594
```

And therefore:

$$H(z) = \frac{2.99144z^3 - 5.29555z^2 + 2.36158z}{1.00000z^3 - 2.62820z^2 + 2.31507z - 0.68594}$$

Using the Matlab function impinvar, the same result can be easily attained using:

```
[ Nz, Dz ] = impinvar(  [ 1 -1 -6], [1 3 5 -1], 0.1 );
```

Solution 7.4.3-1: The design of a filter using the bilinear transformation consists of three steps: prewarping the specs, designing the analog filter and then transforming it into a digital filter using the bilinear transformation.

- The hard way
  1. Prewarping the specs
     We need to prewarp the digital specs into analog specs. This is done using the prewarping function:
     $$\omega_c = \frac{2}{T_s} \tan\left(\frac{\omega_d T_s}{2}\right)$$
     Let's first set the specs in radians and rewrite the sampling frequency as a sampling period:
     $$\omega_{co} = 2\pi f_{co} = 4\pi 1 \times 10^5 \text{ rad}$$
     $$T_s = \frac{1}{f_s} = 1 \times 10^6 \text{ s}$$
     Now, we are ready to prewarp the specs:
     $$\begin{aligned} \omega_c &= \frac{2}{T_s} \tan\left(\frac{\omega_{co} T_s}{2}\right) \\ &= \frac{2}{T_s} \tan\left(\frac{4\pi 10^5 \cdot 110^{-6}}{2}\right) \\ &= \frac{2}{T_s} \tan(0.2\pi) \\ &= \frac{2}{T_s} \underbrace{0.72654}_{\alpha} \end{aligned}$$

Note that we left the coefficent $\frac{2}{T_s}$ intact.

2. Designing the analog filter
   This is most simple:
$$\begin{aligned} H(s) &= \frac{s}{s+\omega_c} \\ &= \frac{s}{s+\frac{2}{T_s}\alpha} \end{aligned}$$

3. Bilinear transformation
$$\begin{aligned} H(s) &= \frac{s}{s+\omega_c} \\ &\Big\downarrow s \mapsto \frac{2}{T_s}\frac{z-1}{z+1} \\ &= \frac{\frac{2}{T_s}\frac{z-1}{z+1}}{\frac{2}{T_s}\frac{z-1}{z+1}+\frac{2}{T_s}\alpha} \\ &\Big\downarrow \text{cancel out } \frac{2}{T_s} \\ &= \frac{\frac{z-1}{z+1}}{\frac{z-1}{z+1}+\alpha} \\ &\downarrow \text{multiply numerator and denominator by } z+1 \\ &= \frac{z-1}{z-1+\alpha(z+1)} \\ &= \frac{z-1}{(1+\alpha)z+(\alpha-1)} \\ &\downarrow \text{divide numerator and denominator by } \alpha+1 \\ &= \frac{\frac{1}{\alpha+1}z-\frac{1}{\alpha+1}}{z+\frac{\alpha-1}{\alpha+1}} \end{aligned}$$

- Using Octave/Matlab

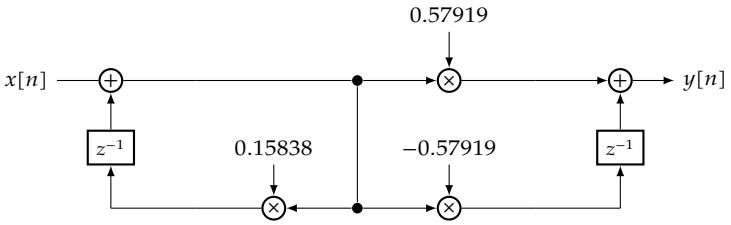
1. Prewarping the specs
```
Ts = 1e-6;
wd = 4e5 * pi;
wc = 2/Ts * tan( wd * Ts / 2 );
```

2. Designing the analog filter
```
b = [ 1 0 ];
a = [ 1 wc ];
```

3. Bilinear transformation
```
[ zb, za ] = bilinear( b, a, Ts )
==> zb = 0.57919  -0.57919
==> za = 1.00000  -0.15838
```

Verifying the result can be done through:
```
[ H, W ] = freqz( zb, za, 1024 );
freqz_plot( W, H );
```

A transposed form I is most easily obtained, by drawing the direct form I, and then transforming it. The result is:



Solution 7.4.3-2: The design of a filter using the bilinear transformation consists of three steps: prewarping the specs, designing the analog filter and then transforming it into a digital filter using the bilinear transformation.

- The hard way
  1. Prewarping the specs
     Let's first rewrite the sampling frequency as a sampling period:
     $$T_s = \frac{1}{f_s} = 1 \times 10^{-3}\,\text{s}$$
     We need to prewarp the digital specs into analog specs. This is done using the prewarping function:
     $$\begin{aligned}\omega_c &= \frac{2}{T_s}\tan\left(\frac{\omega_0 T_s}{2}\right)\\ &= \frac{2}{T_s}\tan\left(\frac{2.410^3 \cdot 110^{-3}}{2}\right)\\ &= \frac{2}{T_s}\tan(1.2)\\ &= \frac{2}{T_s}\underbrace{2.5722}_{\alpha}\end{aligned}$$
     Note that we left the coefficent $\frac{2}{T_s}$ intact.
  2. Designing the analog filter
     This is most simple:
     $$H(s) = \frac{s^2 + \left(\frac{2}{T_s}\alpha\right)^2}{s^2 + \frac{2\alpha}{10T_s}s + \left(\frac{2}{T_s}\alpha\right)^2}$$
  3. Bilinear transformation

$$
\begin{aligned}
H(s) &= \frac{s^2 + \left(\frac{2}{T_s}\alpha\right)^2}{s^2 + \frac{2\alpha}{10T_s}s + \left(\frac{2}{T_s}\alpha\right)^2} \\
&\Big\downarrow s \mapsto \frac{2}{T_s}\frac{z-1}{z+1} \\
&= \frac{\left(\frac{2}{T_s}\frac{z-1}{z+1}\right)^2 + \left(\frac{2}{T_s}\alpha\right)^2}{\left(\frac{2}{T_s}\frac{z-1}{z+1}\right)^2 + \frac{2\alpha}{10T_s}\frac{2}{T_s}\frac{z-1}{z+1} + \left(\frac{2}{T_s}\alpha\right)^2} \\
&\Big\downarrow \text{cancel out } \frac{2}{T_s} \\
&= \frac{\left(\frac{z-1}{z+1}\right)^2 + \alpha^2}{\left(\frac{z-1}{z+1}\right)^2 + \frac{\alpha}{10}\frac{z-1}{z+1} + \alpha^2} \\
&\Big\downarrow \text{multiply numerator and denominator by } (z+1)^2 \\
&= \frac{z-1)^2 + \alpha^2(z+1)^2}{(z-1)^2 + \frac{\alpha}{10}(z^2-1) + \alpha^2(z+1)^2} \\
&= \frac{(\alpha^2+1)z^2 + 2(\alpha^2-1)z + (\alpha^2+1)}{(\alpha^2+\alpha/10+1)z^2 + 2(\alpha^2-1)z + (\alpha^2-\alpha/10+1)} \\
&\Big\downarrow \text{divide numerator and denominator by } \alpha^2+\alpha/10+1 \\
&= \frac{\frac{(\alpha^2+1)}{\alpha^2+\alpha/10+1}z^2 + 2\frac{\alpha^2-1}{\alpha^2+\alpha/10+1}z + \frac{\alpha^2+1}{\alpha^2+\alpha/10+1}}{z^2 + 2\frac{\alpha^2-1}{\alpha^2+\alpha/10+1}z + \frac{\alpha^2-\alpha/10+1}{\alpha^2+\alpha/10+1}}
\end{aligned}
$$

- Using Octave/Matlab
  1. Prewarping the specs

```
Ts = 1e-3;
wd = 2400;
wc = 2/Ts * tan( wd * Ts / 2 );
```

  2. Designing the analog filter

```
b = [ 1   0    wc^2 ];
a = [ 1 wc/10  wc^2 ];
```

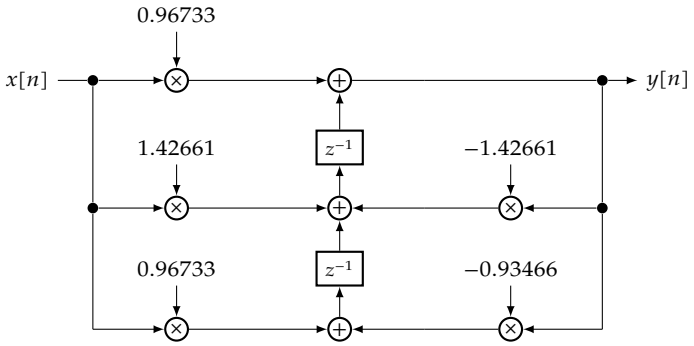
  3. Bilinear transformation

```
[ zb, za ] = bilinear( b, a, Ts )
==> zb = 0.96733   1.42661   0.96733
==> za = 1.00000   1.42661   0.93466
```

Verifying the result can be done through:

```
[ H, W ] = freqz( zb, za, 1024 );
freqz_plot( W, H );
```

A transposed form II is most easily obtained, by drawing the direct form I, morphing it into a direct form II and then transforming it.
The result is:



Solution 7.4.3-3: Let's skip the 'hard way' this time and rely on our mathematical tool to do the hard work for us. Before heating our CPU, let's analyze the problem.
If the pre-emphasis filter has a transfer function $H_p(s)$, then the de-emphasis filter needs to be:

$$H_d(s) = \frac{1}{H_p(s)} = \frac{1 + \tau_2 s}{(1 + \tau_1 s)(1 + \tau_3 s)}$$

Now, let's rework this analog filter, such that frequencies are visible.

$$H(s) = \frac{\tau_2}{\tau_1 \tau_3} \frac{\overbrace{1/\tau_2}^{\omega_2} + s}{(\underbrace{1/\tau_1}_{\omega_1} + s)(\underbrace{1/\tau_3}_{\omega_3} + s)}$$

Realizing that $\omega = 1/\tau$, we can rewrite our prewarping equation:

$$\tau_c = \frac{T_s}{2} \frac{1}{\tan\left(\frac{T_s}{2\tau_d}\right)}$$

1. Prewarping the specs

```
Ts   = 1 / 44.1e3;
taud1 = 3180e-6 / 2 / pi;
taud2 = 318e-6 / 2 / pi;
taud3 = 75e-6 / 2 / pi;
tauc1 = Ts / 2 / tan( Ts / taud1 / 2 );
tauc2 = Ts / 2 / tan( Ts / taud2 / 2 );
tauc3 = Ts / 2 / tan( Ts / taud3 / 2 );
```

2. Designing the analog filter

```
b = [ tauc2 1 ];
a = conv( [ tauc1 1 ], [ tauc3 1 ] );
```

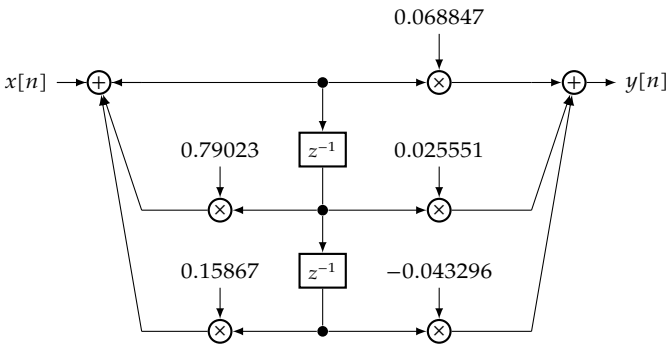
3. Bilinear transformation

```
[ zb, za ] = bilinear( b, a, Ts )
==> zb =  0.068847   0.025551  -0.043296
==> za =  1.00000   -0.79023   -0.15867
```

Verifying the result can be done through:

```
[ H, W ] = freqz( zb, za, 1024 );
figure(1);
freqz_plot( W, H );
```

A direct form II is most easily obtained. The result is:



Now, let's compare the result to a classical de-emphasis filter

```
sb = [ taud2 1 ];
sa = conv( [ taud1 1 ], [ taud3 1 ] );
Ws = linspace( 0, 1 / Ts / 2, 1024 );
Hs = freqs( sb, sa, Ws );
figure(2);
freqs_plot( Ws, Hs );
```

When comparing the two Bode plots, you will see that the result is not terrific. Especially for high frequencies, the deviation is considerable. The conclusion is that the bilinear transformation is only useful for filters with flat bands and sharp transition regions. We'd better select a different filter design method to create a good de-emphasis filter.

Solution 7.5.2-1: The first step is to design a digital IIR filter, starting from the given analog filter. We can use the bilinear transformation to this end. Let's start by prewarping our specs.

```
Ts = 1 / 48e3;
wld = 2 * pi * 20;
whd = 2 * pi * 20e3;
wlc = 2 / Ts * tan( wld * Ts / 2 );
whc = 2 / Ts * tan( whd * Ts / 2 );
```

Then, we can design the filter:

```
a1 = 1 / wlc;
a2 = a1 / whc;
bs = [ a1 0 ];
as = [ a2 a1 1 ];
```

Followed by the third step, the bilinear transformation:

```
[ bz, az ] = bilinear( bs, as, Ts )
==> bz =  0.78786   0.00000  -0.78786
==> az =  1.00000  -0.42015  -0.57572
```

This leads to the following filter:

$$H(z) = \frac{0.78786z^2 - 0.78786}{z^2 - 0.42015z - 0.57572}$$

Now, as a first step, time reversing the data, then filtering, again time reversing the data and again filtering, does the required job.

Chapter 8

Signal Transforms — Short-time Fourier Transform

Solution 8.5-1:

- $f[n] = \begin{bmatrix} 1 & \textcircled{2} & 1 \end{bmatrix}$:
  The function $f[n]$ is symmetric around $n = 0$. That's good. However, the total energy in the signal does not equal 1. Indeed: $\|f[n]\|^2 = 1^2 + 2^2 + 1^2 = 6$. Therefore, it is not a proper window function.
  We can correct it by normalization, i.e. dividing by $\|f[n]\| = \sqrt{6}$, leading to:
  $g_n[n] = \begin{bmatrix} \frac{1}{\sqrt{6}} & \textcircled{\frac{\sqrt{2}}{\sqrt{3}}} & \frac{1}{\sqrt{6}} \end{bmatrix}$.
- $g[n] = \begin{bmatrix} -\frac{1}{\sqrt{6}} & \textcircled{\frac{\sqrt{2}}{\sqrt{3}}} & -\frac{1}{\sqrt{6}} \end{bmatrix}$:
  The function $g[n]$ is symmetric around $n = 0$. That's good. In addition, the total energy in the signal equals 1. Indeed: $\|g[n]\|^2 = \left(\frac{-1}{\sqrt{6}}\right)^2 + \left(\frac{\sqrt{2}}{\sqrt{3}}\right)^2 + \left(\frac{-1}{\sqrt{6}}\right)^2 = 1$. Therefore, it is a proper window function.
- $h[n] = \begin{bmatrix} 0.2 & 0.8 & \textcircled{0.9} & 1.0 & 0.9 & 0.8 & 0.2 \end{bmatrix}$:
  The function $h[n]$ is not symmetric around $n = 0$. However, this is not a fundamental problem. We can correct for this by taking this non-center symmetry point into account when considering its position and its size in time. However, the total energy in the signal does not equal 1. Indeed: $\|h[n]\|^2 = 0.2^2 + 0.8^2 + 0.9^2 + 1^2 + 0.9^2 + 0.8^2 + 0.2^2 = 3.98$. Therefore, it is not a proper window function.
  We can correct it by normalization, leading to:
  $h_n[n] = \begin{bmatrix} 0.10025 & 0.40100 & \textcircled{0.45113} & 0.50125 & 0.45113 & 0.40100 & 0.10025 \end{bmatrix}$.

Solution 8.5-2: If we use our calculator or Octave/Matlab to calculate a Blackman window of length 7 (`>> blackman(7)`), we obtain:

$$b_7[n] = \begin{bmatrix} 0 & 0.13 & 0.63 & \textcircled{1} & 0.63 & 0.13 & 0 \end{bmatrix}]$$

However, this window function is not normalized, it's energy equals:

$$\|b_7[n]\|^2 = 0^2 + 0.13^2 + 0.63^2 + 1^2 + 0.63^2 + 0.13^2 + 0^2 = 1.8276$$

The proper window can be obtained by dividing the result by $\sqrt{1.8276}$, resulting in:

$$w[n] = \frac{1}{\sqrt{1.8276}} b_7[n] = \begin{bmatrix} 0 & 0.096162 & 0.466015 & \textcircled{0.739706} & 0.466015 & 0.096162 & 0 \end{bmatrix}$$

Solution 8.5-3: The following will do:

```
function w = gaborhamming( N )
  w = hamming( N );
  w = w / norm( w );
end
```

Solution 8.5-4: The following does the job:

```
function w = gaborwindow( window, N )
  w = window( N );
  w = w / norm( w );
end
```

Then, the function can be called as:

```
w = gaborwindow( @hamming, N );
```

Solution 8.5.2-1: As $w[n]$ is symmetric w.r.t. the origin, its position is $n_{\eta,\kappa} = \eta$.

Let's calculate its size:

$$\begin{aligned}
\sigma^2_{n,\eta,\kappa} &= sum_n (n - n_{\eta,\kappa})^2 \, |w[n-\eta]|^2 \\
&= sum_n (n-\eta)^2 \, |w[n-\eta]|^2 \\
&= \sum_\nu \nu^2 \, |w[\nu]|^2 \\
&= (-1)^2 \cdot \left(\frac{1}{\sqrt{6}}\right)^2 + 0^2 \cdot \left(-\frac{\sqrt{2}}{\sqrt{3}}\right)^2 + 1^2 \cdot \left(\frac{1}{\sqrt{6}}\right)^2 \\
&= \frac{1}{6} + 0 + \frac{1}{6} = \frac{1}{3}
\end{aligned}$$

The size in time of the atom therefore is: $\sigma_{\eta,\kappa} = \frac{1}{\sqrt{3}}$.

Solution 8.5.2-2: Before we start, we calculate the FFT of the original window function: $w[n] \xrightarrow{\text{DFT}} W[k]$.

We can determine $W[k]$ to be:

$$W[k] = \sum_{n=-1}^{1} w[n] \, \mathrm{e}^{-\mathrm{j}\frac{2\pi}{3}kn} = \begin{bmatrix} 0.40825 & 1.63299 & 0.40825 \end{bmatrix}$$

Can you determine a way to have Octave/Matlab perform this calculation for you?

In any case, its position is fixed: $k_{\eta,\kappa} = \kappa$.

Let's calculate its size:

$$\begin{aligned}
\sigma^2_{k,\eta,\kappa} &= \sum_\beta (k-\kappa)^2 \frac{|W[k-\kappa]|^2}{N} \\
&= \sum_\beta \beta^2 \frac{|W[\beta]|^2}{N} \\
&= (-1)^2 \cdot \frac{(0.40825)^2}{3} + 0^2 \cdot \frac{(1.63299)^2}{3} + 1^2 \cdot \frac{(0.40825)^2}{3} \\
&= 0.11111
\end{aligned}$$

The size in frequency of the atom therefore is: $\sigma_{k,\eta,\kappa} = 0.33333$.

Solution 8.5.2-3: The window can be found to be:

$$w = \begin{bmatrix} \textcircled{0} & 0.22942 & 0.45883 & 0.68825 & 0.45883 & 0.22942 & 0 \end{bmatrix}$$

The position in time can be found to be:

$$\begin{aligned} n_{\eta,\kappa} &= \sum_n n \left|w[n-\eta]\right|^2 \\ &= \sum_\beta (\beta+\eta) \left|w[\beta]\right|^2 \\ &= \sum_\beta \beta \left|w[\beta]\right|^2 + \eta \sum_\beta \left|w[\beta]\right|^2 \\ &= \sum_\beta \beta \left|w[\beta]\right|^2 + \eta \\ &+\eta \\ &= 0\cdot(0)^2 + 1\cdot(0.22942)^2 + 2\cdot(0.45883)^2 + 3\cdot(0.68825)^2 \\ &\quad + 4\cdot(0.45883)^2 + 5\cdot(0.22942)^2 + 6\cdot(0)^2 + \eta \\ &= 3+\eta \end{aligned}$$

which could be expected, because of the symmetry at $n = 3$.

The deriviation made in the text book assumed a symmetrical window. In this case, the window is nonsymmetrical. The size is therefore calculated as:

$$\begin{aligned} \sigma^2_{n,\eta,\kappa} &= \sum_n (n - n_{\eta,\kappa})^2 \left|w[n-\eta]\right|^2 \\ &= \sum_n (n-\eta)^2 \left|w[n-\eta]\right|^2 \\ &= \sum_n (n-(3+\eta))^2 \left|w[n-\eta]\right|^2 \\ &\downarrow \quad \text{substitution: } \nu = n-\eta \\ &= \sum_\nu (\nu-3)^2 \left|w[\nu]\right|^2 \\ &= (0-3)^2\cdot(0)^2 + (1-3)^2\cdot(0.22942)^2 + (2-3)^2\cdot(0.45883)^2 + (3-3)^2\cdot(0.68825)^2 \\ &\quad + (4-3)^2\cdot(0.45883)^2 + (5-3)^2\cdot(0.22942)^2 + (6-3)^2\cdot(0)^2 \\ &= 0.84211 \end{aligned}$$

The size in time of the atom therefore is: $\sigma_{n,\eta,\kappa} = 0.91766$.

Solution 8.5.2-4: The used Bartlett window can be found in the solution of the previous exercise. Before we start, we calculate the FFT of the original window function: $w[n] \xrightarrow{\text{DFT}} W[k]$.

We can determine $W[k]$ to be:

$$\begin{aligned} |W[k]| &= \left| \sum_{n=0}^{6} w[n]\, \mathrm{e}^{-\mathrm{j}\frac{2\pi}{7}kn} \right| \\ &= \begin{bmatrix} 0.147538 & 0.070655 & 1.158301 & \textcircled{2.064742} & 1.158301 & 0.070655 & 0.147538 \end{bmatrix} \end{aligned}$$

Can you determine a way to have Octave/Matlab perform this calculation for you?

In any case, its position is fixed: $k_{\eta,\kappa} = \kappa$.

Let's calculate its size:

$$\begin{aligned}\sigma^2_{k,\eta,\kappa} &= \sum_{\beta} \beta^2 \frac{|W[\beta]|^2}{N} \\ &= \frac{1}{7}\Big((-3)^2\cdot(0.147538)^2 + (-2)^2\cdot(0.070655)^2 + (-1)^2\cdot(1.158301)^2 + (0)^2\cdot(2.064742)^2 \\ &\qquad +(1)^2\cdot(1.158301)^2 + (2)^2\cdot(0.070655)^2 + (3)^2\cdot(0.147538)^2\Big) \\ &= 0.44501\end{aligned}$$

The size in frequency of the atom therefore is: $\sigma_{k,\eta,\kappa} = 0.66709$.

Solution 8.5.2-5: The following function does the job:

```
function [ n, sigma ] = atomtimespecs( w )
  %% 'atomtimespecs' calculates the position and size in time of a
  %% Gabor atom with a proper window w.
  %%
  %% PARAMETERS: (w)
  %% w: window vector
  %% - must be a column vector
  %% - will be normalized
  %% - the window is assumed to be right-winged (i.e. causal)
  %%
  %% RETURN VALUE: [pos, size]
  %% n    : net position of the window in time (i.e. position -
  %% time shift value)
  %% sigma: size of the window in time
  %%
  %% (C) 2016 - W. Daems

  [N,M] = size(w);
  assert( M == 1, 'atomtimespecs(␣w␣):␣w␣must␣be␣a␣column␣vector' );

  w = w / sqrt( sumsq( w ) );

  Nminusone = N - 1;

  n = Nminusone / 2;
  sigma = sqrt( (-n:n).^2 * w.^2 );
end
```

Note that the function also works for window functions of odd length.

Solution 8.5.2-6: The following function does the job:

```
function [ k, sigma ] = atomfreqspecs( w )
  %% 'atomfreqspecs' calculates the position and size in frequency of a
  %% Gabor atom with a proper window w.
  %%
  %% PARAMETERS: (w)
  %% w: window vector
  %%    - must be a column vector
  %%    - will be normalized
  %% - the window is assumed to be right-winged (i.e. causal)
  %%
  %% RETURN VALUE: [pos, size]
  %% k    : net position of the window in frequency (i.e. position -
  %% time shift value)
  %% sigma: size of the window in frequency
  %%
  %% (C) 2016 - W. Daems

  [N,M] = size(w);
  assert( M == 1, 'atomfreqspecs(␣w␣):␣w␣must␣be␣a␣column␣vector' );

  w = w / sqrt( sumsq( w ) );
  W = fftshift( abs( fft( w ) ) );

  Nminusone = N-1;
  halfN = floor(N/2);
```

```
  k = 0
  sigma = sqrt( ( (0:Nminusone) - halfN ).^2 * W.^2 / N );
end
```

Solution 8.5.2-7: The following function does the job:

```
function [ w, tpos, tsize, fpos, fsize ] = gaborwindow( window, N )
  w = gaborwindow( window, N );
  [tpos, tsize] = atomtimespecs( w );
  [fpos, fsize] = atomfreqspecs( w );
end
```

Solution 8.6.1-1: The window had a length of 256 (given the fact that □ranges from 0 to 255).
There are 44100 time points (n ranges from 0 to 44099), therefore, the sampling frequency is 44.1 kHz.
The sine wave lasts from 0.25 s to 0.75 s.
The linear chirp goes up in frequency starting at $\kappa \approx 5$ and ending at $\kappa \approx 24$. These values of κ correspond to frequencies of 861 Hz and 4 134 Hz respectively.
The quadratic chirp goes down in frequency starting at $\kappa \approx 96$ and ending at $\kappa \approx 5$. These values of κ correspond to frequencies of 16.5 kHz and 861 Hz respectively.
The obtained values of κ were converted to the corresponding frequencies, using:

$$f = \frac{\kappa}{N} f_s = \frac{\kappa}{256} 44.1\,\text{kHz}$$

The value of $\kappa = 192$ corresponds to $f_{192,\text{alias}} = 33.075\,\text{kHz}$. However, this frequency is an alias of $f_{192} = f_{192,\text{alias}} - f_s = -11.025\,\text{kHz}$.

Solution 8.6.1-2: We can apply zero-padding to (both sides of) the window function. This will not increase the size of the atom in the time-domain, but it will increase the frequency resolution.

Chapter 9

Signal Transforms — Wavelets

Solution 9.1-1: The scale from A to a takes twelve half-tone steps. The full octave step (from A to a) has a step height of 2:

$$f_a = 2f_A$$

Let's call the half-tone step of the equal tempering h, and lets relabel the frequency f_A as f_0, $f_{A\sharp}$ as f_1, f_B as f_2, continuing to f_a as f_{12}. This allows writing in general:

$$f_{n+1} = hf_n$$

And by applyling the induction in particular:

$$\begin{aligned} f_1 &= hf_0 \\ f_2 &= h^2 f_0 \\ f_3 &= h^3 f_0 \\ &\dots \\ f_{12} &= h^{12} f_0 \end{aligned}$$

Considering the first equation of this solution and the latest equation, leads to:

$$h^{12} = 2$$

and therefore:

$$h = \sqrt[12]{2}$$

This allows deriving e.g.:

$$f_c = f_3 = h^3 f_0 = 2^{\frac{3}{12}} 440\,\text{Hz} = 523.25\,\text{Hz}$$

Solution 9.1-2: Show your solution to the lecturer. He'll judge its quality.

Solution 9.3.4-1:

$$\begin{aligned} \vec{c} = [&-0.7071, \quad 0.7071, \quad 4.9497, \quad 1.4142, \quad 0.7071, \quad -2.1213, \quad | \\ &-3.5355, \quad -0.7071, \quad -0.7071, \quad 1.4142, \quad -0.7071, \quad 0.7071] \end{aligned}$$

Solution 9.3.4-2:

$$\vec{c} = \left[-0.7071, \quad -3.5355, \quad 1.4142, \quad 1.4142, \quad 0, \quad 0, \quad 1.4142, \quad 0, \quad 1.4142, \quad 2.8284, \quad 1.4142, \quad 4.2426\right]$$

Solution 9.3.5-1:

$$\vec{c}_3 = \left[3.1820, \quad -3.1820, \quad -1.0000, \quad 2.5000, \quad -3.5355, \quad -0.7071, \quad -0.7071, \quad 1.4142\right]$$

Solution 9.3.5-2:

$$\vec{c} = \left[1.7678, \quad -2.4749, \quad 2.4749, \quad -3.1820, \quad 0.1464, \quad -2.6820, \quad -2.2678, \quad -2.2678\right]$$

Solution 9.3.6-1: The index border in between scaling an wavelet signals for a 1-level Haar transform is:

$$M_1 = \frac{N}{2^1} = \frac{64}{2} = 32$$

Therefore, $\vec{s}_{1,17}$ is a valid scaling signal descriptor.

We apply the recursion definition for Haar base vectors:

$$\vec{s}_{n,i} = \sum_{k=0}^{1} \alpha_k \vec{s}_{n-1,2i+k}$$

In this case, $n = 1$ and $i = 17$. Therefore:

$$\begin{aligned}\vec{s}_{1,17} &= \alpha_0 \vec{s}_{0,34} + \alpha_1 \vec{s}_{0,35} \\ &\downarrow \quad \vec{s}_{0,j} = \delta[n-j] \text{ and } \alpha_0 = \alpha_1 = 1/\sqrt{2} \\ &= \frac{1}{\sqrt{2}}\delta[n-34] + \frac{1}{\sqrt{2}}\delta[n-35]\end{aligned}$$

Solution 9.3.6-2: First, we try to find the largest possible border M_n that is smaller or equal to the wavelet base vector number, i.e. 35. For $N = 64$, that is:

$$M_1 = \frac{N}{2^1} = \frac{64}{2} = 32 \leq 35 < M_0 = 64$$

Therefore, $\vec{w}_{35}$ is a first-level wavelet base vector.

Starting from the generic recursion equation for wavelet base vectors

$$\vec{w}_{M_n+i} = \sum_{k=0}^{1} \beta_k s_{n-1,2i+k}$$

and noting that $i = 35 - M_1 = 3$ and $n = 1$, we obtain:

$$\begin{aligned}\vec{w}_{35} &= \beta_0 \vec{s}_{0,6} + \beta_1 \vec{s}_{0,7} \\ &\downarrow \quad \vec{s}_{0,j} = \delta[n-j] \text{ and } \beta_0 = -\beta_1 = 1/\sqrt{2} \\ &= \frac{1}{\sqrt{2}}\delta[n-6] - \frac{1}{\sqrt{2}}\delta[n-7]\end{aligned}$$

Solution 9.3.6-3: The index border in between scaling an wavelet signals for a 4-level Haar transform is:

$$M_4 = \frac{N}{2^4} = \frac{64}{16} = 4$$

Therefore, $\vec{s}_{4,3}$ is a valid scaling signal descriptor.

We apply the recursion definition for Haar base vectors:

$$\vec{s}_{n,i} = \sum_{k=0}^{1} \alpha_k \vec{s}_{n-1,2i+k}$$

In this case, $n = 4$ and $i = 3$. Therefore:

$$\begin{aligned}
\vec{s}_{4,3} &= \alpha_0 \vec{s}_{3,6} + \alpha_1 \vec{s}_{3,7} \\
&= \alpha_0 \left(\alpha_0 \vec{s}_{2,12} + \alpha_1 \vec{s}_{2,13}\right) + \alpha_1 \left(\alpha_0 \vec{s}_{2,14} + \alpha_1 \vec{s}_{2,15}\right) \\
&= \alpha_0 \left(\alpha_0 \left(\alpha_0 \vec{s}_{1,24} + \alpha_1 \vec{s}_{1,25}\right) + \alpha_1 \left(\alpha_0 \vec{s}_{1,26} + \alpha_0 \vec{s}_{1,27}\right)\right) \\
&\quad + \alpha_1 \left(\alpha_0 \left(\alpha_0 \vec{s}_{1,28} + \alpha_1 \vec{s}_{1,29}\right) + \alpha_1 \left(\alpha_0 \vec{s}_{1,30} + \alpha_1 \vec{s}_{1,31}\right)\right) \\
&= \alpha_0 \left(\alpha_0 \left(\alpha_0 \left(\alpha_0 \delta[n-48] + \alpha_1 \delta[n-49]\right) + \alpha_1 \left(\alpha_0 \delta[n-50] + \alpha_1 \delta[n-51]\right)\right)\right. \\
&\qquad \left. + \alpha_1 \left(\alpha_0 \left(\alpha_0 \delta[n-52] + \alpha_1 \delta[n-53]\right) + \alpha_0 \left(\alpha_0 \delta[n-54] + \alpha_1 \delta[n-55]\right)\right)\right) \\
&\quad + \alpha_1 \left(\alpha_0 \left(\alpha_0 \left(\alpha_0 \delta[n-56] + \alpha_1 \delta[n-57]\right) + \alpha_0 \left(\alpha_0 \delta[n-58] + \alpha_1 \delta[n-59]\right)\right)\right. \\
&\qquad \left. + \alpha_1 \left(\alpha_0 \left(\alpha_0 \delta[n-60] + \alpha_1 \delta[n-61]\right) + \alpha_0 \left(\alpha_0 \delta[n-62] + \alpha_1 \delta[n-63]\right)\right)\right) \\
&= \frac{1}{4} \sum_{i=48}^{63} \delta[n-i]
\end{aligned}$$

Solution 9.3.6-4: First, we try to find the largest possible border M_n that is smaller or equal to the wavelet base vector number, i.e. 9. For $N = 64$, that is:

$$M_3 = \frac{N}{2^3} = \frac{64}{8} = 8 \leq 9 < M_2 = 16$$

Therefore, $\vec{w}_9$ is a third-level wavelet base vector.

Starting from the generic recursion equation for wavelet base vectors

$$\vec{w}_{M_n+i} = \sum_{k=0}^{1} \beta_k s_{n-1,2i+k}$$

and noting that $i = 9 - M_3 = 1$ and $n = 3$, we obtain:

$$\begin{aligned}
\vec{w}_9 &= \beta_0 \vec{s}_{2,2} + \beta_1 \vec{s}_{2,3} \\
&= \beta_0 \left(\left(\alpha_0 \vec{s}_{1,4} + \alpha_1 \vec{s}_{1,5}\right)\right) + \beta_1 \left(\left(\alpha_0 \vec{s}_{1,6} + \alpha_1 \vec{s}_{1,7}\right)\right) \\
&= \beta_0 \left(\alpha_0 \left(\alpha_0 \delta[n-8] + \alpha_1 \delta[n-9]\right) + \alpha_1 \left(\alpha_0 \delta[n-10] + \alpha_1 \delta[n-11]\right)\right) \\
&\quad + \beta_1 \left(\alpha_0 \left(\alpha_0 \delta[n-12] + \alpha_1 \delta[n-13]\right) + \alpha_1 \left(\alpha_0 \delta[n-14] + \alpha_1 \delta[n-15]\right)\right) \\
&= \frac{1}{2\sqrt{2}} \sum_{i=8}^{11} \delta[n-i] - \frac{1}{2\sqrt{2}} \sum_{i=12}^{15} \delta[n-i]
\end{aligned}$$

Solution 9.3.6-5: As $M_2 = N/2^2 = 16 < 20$, the base vector $\vec{s}_{2,20}$ is not a proper base vector of a wavelet transform with $N = 64$. Conclusion: there is an error in the indices. The question is invalid.

Solution 9.10.1-1:

1. The entropy can be calculated as:

$$\begin{aligned}
H(\vec{f}) &= \frac{1}{8}\log_2(8) + \frac{1}{8}\log_2(8) + \frac{1}{8}\log_2(8) + \frac{1}{8}\log_2(8) + \frac{1}{8}\log_2(8) + \frac{1}{8}\log_2(8) + \frac{1}{8}\log_2(8) + \frac{1}{8}\log_2(8) \\
&= 3\,\text{bit}
\end{aligned}$$

   It seems obvious that an eight-value code that has equal occurrence frequencies, results in a 3-bit encoding, since $2^3 = 8$.
2. One needs 24 bit to encode this signal.

Solution 9.10.1-2:

1. The entropy can be calculated as:

$$\begin{aligned}
H(\vec{f}) &= 1\log_2(1) + 1\log_2(1) + 1\log_2(1) + 1\log_2(1) + 1\log_2(1) + 1\log_2(1) + 1\log_2(1) + 1\log_2(1) \\
&= 0\,\text{bit}
\end{aligned}$$

2. One needs 0 bit to encode this signal.

Solution 9.10.1-3:

1. The entropy can be calculated as:

$$\begin{aligned} H(\vec{f}) &= \frac{1}{8}\log_2 8 + \frac{1}{4}\log_2 4 + \frac{1}{2}\log_2 2 + \frac{1}{8}\log_2 8 \\ &= 1.75\,\text{bit} \end{aligned}$$

2. One needs 14 bit to encode this signal.

Solution 9.10.1-4:

1. The entropy can be calculated as:

$$\begin{aligned} H(\vec{f}) &= \frac{1}{8}\log_2 8 + \frac{1}{4}\log_2 4 + \frac{1}{2}\log_2 2 + \frac{1}{8}\log_2 8 \\ &= 1.75\,\text{bit} \end{aligned}$$

   It shouldn't be suprising that the result is the same as for the previous exercise. Indeed, the entropy is not dependent on the specific symbols used for the encoding.
2. One needs 14 bit to encode this signal.