The DSP Workbook

A companion to EE 463

Version 1.0
Spring 2011

Matthew C. Valenti, Ph.D.
West Virginia University

Preface

This workbook is based on notes used by the author while teaching EE 463, Digital Signal Processing Fundamentals, at West Virginia University each Spring semester from 2006 to 2010. The book draws heavily from two textbooks used during those semester:

1. S. K. Mitra, Digital Signal Processing: A Computer-Based Approach, Third Edition, McGraw Hill, 2006.

2. V. K. Ingle and J. G. Proakis, Digital Signal Processing using Matlab, Second Edition, Thompson, 2007.

In 2011, new editions of both books became available. Many of the exercises and problems in this work book are modified versions of those found in the above two references. Furthermore, most of the Matlab functions in this workbook may be found in the Ingle and Proakis book.

The purpose of this workbook is not to duplicate material found in the course texts, but rather to provide a guide to the in-class lecture. It is expected that both the instructor and student would bring a hard copy of this workbook to class. During the class, the instructor should cover the examples and exercises in the workbook, while the student should follow along. The workbook makes heavy use of Matlab, and the class is best taught in a computer classroom so that each student can try the exercises on their own using Matlab. You will notice that there are a lot of blank spaces in the workbook. This is intentional: The student should write the solutions to the exercises in those spaces.

This book is still a work in progress. There will certainly be several typos or other mistakes. Catching a mistake is a good sign that you are paying attention! If you find a mistake in this book, please let me know by emailing me at mvalenti@wvu.edu.

Contents

Chapter 1

Discrete-Time Signal Operators

1.1 Analog-to-Digital Conversion

This course is concerned with the digital processing of signals. If the signal to be processed is analog, then it must first be converted into digital form before processing by using an *analog-to-digital converter* (ADC). After digital processing, the signal needs to be brought back into analog form by using a *digital-to-analog converter* (DAC). Fortunately, most commercial DSP devices (chips) have an on-board ADC and DAC. **Draw a schematic of a basic DSP system here, making sure to include the ADC and DAC:**

Before continuing, it should be pointed out that an ADC actually performs *two* operations:

1. **Sampling:** The analog signal $x(t)$ is a function of the continuous-time variable t which does not lend itself to a digital representation. The first step in the ADC process is to transform $x(t)$ into the discrete-time signal $x[n]$. This process is called *sampling*. The sampling process is reversible if the signal is *bandlimited* and the sample rate is sufficiently high (according to the Nyquist sampling theorem, which is discussed in the next section).

2. **Quantization:** Although discrete in time, the samples $x[n]$ are still continuous in amplitude. To represent digitally, the samples must be *quantized* by rounding them to one of a finite set of *quantization levels* $\{x_q\}$. If there are Q quantization levels, then each sample may be represented by a $\log_2(Q)$ bit quantity. Due to roundoff errors, quantization introduces distortion that cannot be recovered in the DAC process.

An example of sampling and quantization will be presented in the next section once we establish some more notation and terminology.

1.1.1 Discrete-time Signals

Let $x(t)$ be a continuous-time (analog) signal with *bandwidth* W.[1] If $x(t)$ is sampled at a *rate* of F_T samples per second (or Hz), then the resulting discrete-time signal is:

$$x[n] \quad = \quad x(nT) \tag{1.1}$$

where

$$T \quad = \quad \frac{1}{F_T} \tag{1.2}$$

is the *sample period* and $n \in \mathbb{Z}$, that is n belongs to the set of all integers $\{..., -1, 0, +1, +2, ...\}$. Notice that since there are an infinite number of integer sample instances n, then the number of samples is infinite (at least in theory). However, in practice, we must restrict ourselves to a finite number of samples, which implies that n will be a finite set, for instance,

$$n \quad \in \quad \{n_1, n_1+1, ..., n_2-1, n_2\} \tag{1.3}$$

where n_1 is the first sample instance and n_2 is the last sample instance. The length of the sequence is $n_2 - n_1 + 1$. As we will see shortly, when we represent $x[n]$ in Matlab, not only do we need to indicate the values of the samples $x[n]$ themselves, but we also need to indicate the indices n for which the samples were evaluated.

Exercise 1.1: Suppose that the signal is $x(t) = \frac{7}{2}\cos(2\pi t)$. The signal is sampled at a *rate* of $F_T = 8$ Hz. Determine the set of samples $x[n]$ at sample instances $0 \leq n \leq 4$. Represent the samples using a *stem diagram.*

Exercise 1.2: Now suppose that $x[n]$ is passed through a 3-bit quantizer whose maximum level matches the maximum value of the signal (so that there is no *clipping* of the sampled signal). Determine the value of the samples.

While quantization is an important issue, we will ignore it for most of this workbook and just consider the behavior of the signal $x[n]$, which is discrete in time but continuous in amplitude.

[1]The *bandwidth* of a signal is the range of frequencies for which the energy content is non-negligible.

1.1.2 Representing Signals in Matlab

To represent a signal in Matlab, you need to first create the set of sample instances $\{n_1, ..., n_2\}$. This set can be placed into a row vector $\mathbf{n}$. Next, you need to specify the sample rate F_T or equivalently the sample period T. Finally, the signal can be sampled by evaluating it at $x(nT)$ for each n in the vector $\mathbf{n}$. The resulting samples are placed into the vector $\mathbf{x}$, which has the same length as $\mathbf{n}$. Here is exercise 1.1 done in Matlab:

```
n = -10:10;   % create a vector of sample indices
F_T = 8;      % the sample rate
T = 1/F_T;    % the sample period
x = cos( 2*pi*n*T ); % the sampled signal
stem( n, x );    % display using a stem diagram
xlabel( 'n' );   % mark the x-axis
ylabel( 'x[n]' );% mark the y-axis
```

1.1.3 The Nyquist Sampling Theorem

The signal $x(t)$ can be unambiguously recovered from $x[n]$ if $F_T \geq 2W$. Otherwise *aliasing* will occur.

Exercise 1.3: Sample the signal $\cos(2\pi t)$ at a rate of $F_T = 3/2$ Hz. Use Matlab.

Exercise 1.4: Now sample the signal $\cos(\pi t)$ at a rate of $F_T = 2/3$ Hz. Again, use Matlab.

1.2 Elementary Sequences

The following sequences will be used throughout the workbook, and are useful for illustrating key concepts.

1.2.1 Unit-Sample Sequence

The unit-sample sequence $\delta[n]$ is defined as:

$$\delta[n] = \begin{cases} 1 & \text{for } n = 0 \\ 0 & \text{for } n \neq 0 \end{cases} \tag{1.4}$$

where n runs from n_1 to n_2. This signal can be created in Matlab as follows (in this example, $n_1 = -10$ and $n_2 = +10$)

```
n = -10:10;   % create a vector of sample indices
x = (n==0);   % creates a '1' where n=0, and a '0' elsewhere
stem( n, x );    % display using a stem diagram
xlabel( 'n' );   % mark the x-axis
ylabel( 'x[n]' );% mark the y-axis
```

This signal is also called an *impulse* sequence.

1.2.2 Unit-Step Sequence

The unit-step sequence $u[n]$ is defined as:

$$u[n] = \begin{cases} 1 & \text{for } n \geq 0 \\ 0 & \text{for } n < 0 \end{cases} \tag{1.5}$$

where n again runs from n_1 to n_2. This signal can be created in Matlab as follows (in this example, $n_1 = -10$ and $n_2 = +10$)

```
n = -10:10;   % create a vector of sample indices
x = (n>=0);   % creates a '1' where n>=0, and a '0' elsewhere
stem( n, x );    % display using a stem diagram
xlabel( 'n' );   % mark the x-axis
ylabel( 'x[n]' );% mark the y-axis
```

1.3 Implementing Functions in Matlab

Throughout this course, you will be asked to create your own Matlab functions. A function is an entity that takes in one or more *inputs* and produces one or more *outputs*. To start, let's create a function for generating an impulse (type "edit" in Matlab to bring up the editor). The function needs to be stored as its own file (with a ".m" extension). Create a file called "impseq.m" with the following contents:

```
function [x,n] = impseq( n1, n2)
% Generates x[n] = delta[n]
% and the indices n = [n1, ..., n2]
% note that n2 > n1 for this to work
n = [n1:n2];
x = (n==0);
```

Notice that the function takes in two inputs: n_1 and $n2$, and that it creates two outputs, the vector $\mathbf{x}$ containing the sample values and the vector $\mathbf{n}$ containing the sample instances. Back in the Matlab command space, the function may be called as follows:

```
n1 = -10;  % first sample instant
n2 = +10;  % last sample instant
[x,n] = impseq( n1, n2 );  % the impulse sequence
stem( n, x );    % display using a stem diagram
xlabel( 'n' );   % mark the x-axis
ylabel( 'x[n]' );% mark the y-axis
```

Exercise 1.5: Create a function that generates a unit-step sequence. Call the function "stepseq".

1.4 Unary Signal Operations

Unary operations are operations that work on just one signal. Later, we will discuss *binary* operations which act on two signals (not to be confused with *binary* number systems!), but for now, let's go over some basic unary signal operations.

1.4.1 Shifting

One of the most basic operations in a DSP system is to *shift* the time reference. The shift may be considered either a *delay* or an *advance*. Delays are more common than advances, especially if the processing must be done in real-time and must be causal. A signal $y[n]$ may be defined to be $x[n]$ delayed by n_0 samples as follows:

$$y[n] \quad = \quad x[n - n_0] \tag{1.6}$$

If $n_0 > 0$, the signal has been delayed, while if $n_0 < 0$ the signal has been advanced.
Exercise 1.6: Prove the following expression for a time-shifted unit-sample sequence:

$$\delta[n - n_0] \quad = \quad \begin{cases} 1 & \text{for } n = n_0 \\ 0 & \text{for } n \neq n_0. \end{cases} \tag{1.7}$$

We can use this to modify our "impseq" program so that it now takes three arguments, with n_0 being a new argument:

```
function [x,n] = impseq( n0, n1, n2)
% Generates x[n] = delta[n-n0]
% and the indices n = [n1, ..., n2]
% note that n2 > n1 for this to work
n = [n1:n2];
x = (n==n0);
```

Similarly, it can be shown that:

$$u[n - n_0] \quad = \quad \begin{cases} 1 & \text{for } n \geq n_0 \\ 0 & \text{for } n < n_0. \end{cases} \tag{1.8}$$

and a similar change may be done to the "stepseq" function.

More generally, the signal $x[n - n_0]$ is identical to the signal $x[n]$ except that the sample that was at $n = 0$ is now shifted to be at $n = n_0$. The following Matlab function will delay any sequence by n_0 samples:

```
function [y,n] = sigshift(x,n,n0)
% Generates y[n] = x[n-n0]
n = n + n_0; % new time axis
y = x; % the sample sequence remains untouched
```

Exercise 1.7: Let $x[n] = \cos(\pi n/8)$ for $-16 \leq n \leq 16$. On the same stem diagram, plot $x[n]$ and $x[n-4]$. You should use the *sigshift* function to complete this exercise.

1.4.2 Folding

Folding involves the reversal of the time axis. So instead of the signal indices running from n_1 to n_2, they now run from $-n_2$ to $-n_1$. This is written as:

$$y[n] = x[-n] \tag{1.9}$$

In Matlab, this can be implemented with the help of Matlab's "fliplr" function. The following function performs a folding operation:

```
function [y,n] = sigfold(x,n)
% implements y(n) = x(-n)
% -----------------------
% [y,n] = sigfold(x,n)
%
n = -fliplr(n);  % reverse the time axis
y = fliplr(x);   % sample values are unchanged
```

Exercise 1.8: Let $x[n] = \exp(n/5)$ for $-5 \leq n \leq 15$. Plot $x[n]$ and $y[n] = x[-n]$ on the same stem diagram.

1.4.3 Scalar Multiplication

We can multiply a signal by a scalar value such as 'c', which is represented by:

$$y[n] \quad = \quad cx[n] \tag{1.10}$$

In Matlab, multiplication is performed with the asterisk key '$*$', i.e.

```
y = c*x;
```

Multiplication does not change the time axis. Note that later in the notes, we will use '$*$' to denote convolution, so you need to be mindful of this distinction.

Exercise 1.9: For the signal $x[n]$ defined in Exercise 1.8, plot $y[n] = 4x[n]$. Use Matlab.

1.4.4 Exponentiation

In some cases, we might want to *square* each sample in a sequence, resulting in a new sequence. The notation

$$y[n] \quad = \quad (x[n])^2 \tag{1.11}$$

is interpreted as follows: The n^{th} sample of the signal $y[n]$ is found by squaring the sample $x[n]$. More generally, we can raise a signal to the ℓ^{th} power, i.e. $y[n] = (x[n])^{\ell}$. In Matlab, signal exponentiation is performed using

```
y = x.^2;
```

Exercise 1.10: Using Matlab, plot $y[n] = (x[n])^2$, where $x[n] = \cos(\pi n/8)$ and $-16 \leq n \leq 16$.

1.4.5 Sample Summation

Another basic operation is to add up a sequence of consecutive samples. For instance, we could add samples with time indices ranging from n_1 to n_2 as follows:

$$c = \sum_{n=n_1}^{n_2} x[n] = x[n_1] + x[n_1+1] + ... + x[n_2]. \tag{1.12}$$

Note that the result of this type of *sample summation* is a scalar value, not a signal (i.e. the time index is lost since we've summed over it). Summation is performed in Matlab using the "sum" function. For instance,

```
c = sum(x)
```

will give the sum of all the samples stored in the vector **x**.
Exercise 1.11: Use Matlab to find the sum of all samples of the $y[n]$ found in Exercise 1.10.

Now find the sum of all samples with nonnegative index (i.e. all $y[n]$ for which $n \geq 0$).

1.4.6 Signal Energy

The *energy* of a signal $x[n]$ that is defined over indices ranging from n_1 to n_2 is found by performing a sample summation of the signal $|x[n]|^2$ i.e.

$$\mathcal{E}_x = \sum_{n=n_1}^{n_2} |x[n]|^2. \tag{1.13}$$

Exercise 1.12: Use Matlab to find the energy of $x[n] = \cos(\pi n/8)$ and $-16 \leq n \leq 16$.

1.4.7 Signal Power

The *power* of a signal $x[n]$ that is defined over indices ranging from n_1 to n_2 is found by dividing the energy by the number of samples $n_2 - n_1 + 1$, i.e.

$$\mathcal{P}_x = \frac{\mathcal{E}_x}{n_2 - n_1 + 1}. \tag{1.14}$$

If the signal duration is infinite, then this becomes a limiting operation,

$$\mathcal{P}_x = \lim_{K\to\infty} \frac{1}{2K+1} \sum_{n=-K}^{K} |x[n]|^2. \tag{1.15}$$

If the signal is periodic with period N then

$$\mathcal{P}_x = \lim_{K\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} |x[n]|^2 \tag{1.16}$$

where the summation may be over any one period.
Exercise 1.13: Use Matlab to find the power of $x[n] = \cos(\pi n/8)$.

1.4.8 Accumulation

We can use the idea of sample summation to create a signal if we allow the limits to depend on n. An *accumulator* performs sample summation with $n_1 = -\infty$ and $n_2 = n$. This relationship can be expressed as:

$$y[n] = \sum_{\ell=-\infty}^{n} x[\ell]. \tag{1.17}$$

The result is now a signal $y[n]$ because the upper limit in the summation depends on n. Notice that since n is now in the limits of the summation, we need to use a different index of summation (in this case, ℓ).

Now perform a change of variables, letting $\ell = n - k$. This gives an alternative expression:

$$y[n] = \sum_{k=0}^{\infty} x[n-k]. \tag{1.18}$$

The interpretation of this equation is that $y[n]$ is equal to the sum of the current sample $x[n]$ and *all* past samples.

Recursive Representation of Accumulation

Notice that the two equations (1.17) and (1.18) are *equivalent*. Thus, it is apparent that a given signal operation can be represented in more than one way. Are there any other ways to represent the accumulation operation? It turns out that another way is:

$$y[n] \quad = \quad x[n] + y[n-1] \tag{1.19}$$

Proof: Rewrite (1.18) using m as the index of summation:

$$y[n] \quad = \quad \sum_{m=0}^{\infty} x[n-m]. \tag{1.20}$$

Now delay both sides by one sample period:

$$\begin{aligned} y[n-1] \quad &= \quad \sum_{m=0}^{\infty} x[(n-1)-m] \\ &= \quad \sum_{m=0}^{\infty} x[n-(m+1)] \end{aligned} \tag{1.21}$$

Now change the index of summation to $k = m + 1$,

$$y[n-1] \quad = \quad \sum_{k=1}^{\infty} x[n-k]. \tag{1.22}$$

Finally, subtract (1.22) from (1.18),

$$\begin{aligned} y[n] - y[n-1] \quad &= \quad \sum_{k=0}^{\infty} x[n-k] - \sum_{k=1}^{\infty} x[n-k] \\ &= \quad x[n] \end{aligned} \tag{1.23}$$

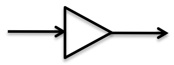
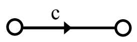
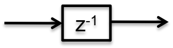
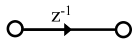
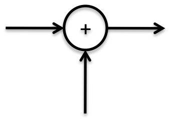
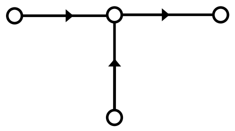
where the second line follows from the fact that the only difference between the two summations is that the first summation has $k = 0$ term but the second one does not. Now, solve (1.23) for $y[n]$ to get:

$$y[n] \quad = \quad x[n] + y[n-1] \tag{1.24}$$

This expression is said to be *recursive* because each output depends on past outputs. An *accumulator* is one of the simplest types of *Infinite Impulse Response* (IIR) filters.

Pictorial Representation of Accumulation

Most signal operations of interest can be represented with the help of *block diagrams*. Alternatively, *signal-flow diagram* may be used. The basic elements for these diagrams are pictured below:

Component	Block Diagram	Signal-flow Diagram
Multiplier		
Delay		
Adder		

To implement an accumulator, we only need two types of elements: An *adder* and a *unit delay*. The multiplier will prove useful for other systems.

Exercise 1.14: Draw block-diagram and signal-flow diagram representations of equation (1.24).

Implementing an Accumulator in Matlab

Now, let's develop an accumulator in Matlab. We will basically be implementing equation (1.24) as a function. One issue we need to deal with is that, because of the recursive nature of the system, the length of the output could be infinite even if the length of the input is finite. Because of this, the system is said to have an *infinite impulse response* and is classified as an *IIR* system. To handle this problem, we will limit the length of the output to be the same as the input. If we want to make the output longer, we can extend the length of the input by padding it with zeros (i.e. just append a sequence of zeros at the end of the sequence).

Here is a function that will perform accumulation:

```
function [y,m] = accumulate(x,n)
% implements y(n) = x(n) + y(n-1)
% ------------------------
% [y,m] = accumulate(x,n)
%
m = n; % no change in time axis
y = zeros( size(x) ); % need to preallocate y
y(1) = x(1); % first output sample
for index=2:length(n)
   y(index) = x(index) + y(index-1);
end
```

Exercise 1.15: Let $x[n] = \delta[n]$ over $-5 \le n \le 15$. Using the "accumulate" function, plot $y[n] = x[n]+y[n-1]$ over the same range of n. What is the relationship among $\delta[n]$, $u[n]$, and accumulation?

Repeat this exercise for $x[n] = u[n]$ and comment about the stability of the accumulation function.

1.4.9 Moving Average

Another way to create a signal using the idea of sample summation is to take the average of M samples (the current sample and past $M-1$ samples). This is an *M-point* average, and may be written as:

$$y[n] \quad = \quad \frac{1}{M}\sum_{k=0}^{M-1} x[n-k] \tag{1.25}$$

This is also called a *moving average* and is one of the simplest types of *Finite Impulse Response* (FIR) filters. Notice that the time averaging operation given in (1.25) is similar to the accumulator as given in (1.18) except that the expressions are different in two ways: What are they?

Pictorial Representation of a Moving Average

Besides adding and delaying signals, we also need the ability to perform a scalar multiplication operation, which can be represented pictorially with a *multiplier*.

Exercise 1.16: Draw block-diagram and signal-flow diagram representations of equation (1.25) for the case that $M = 4$.

1.5 Binary Signal Operations

We would also like to perform operations on two signals, such as addition and multiplication of signals. Such operations are called *binary* because they have two arguments. This section describes some basic binary operations. A more complicated binary signal operation is *discrete-time convolution* which is described in the next chapter.

1.5.1 Addition

Addition of two signals implies that samples at the same time instant are added together. As an example:

$$y[n] \quad = \quad x_1[n] + x_2[n] \tag{1.26}$$

Adding two signals defined over the same time scale is easy in Matlab. For instance:

```
n = 0:12;
x1 = exp(-n/4);
x2= cos(pi*n/4);
y = x1 + x2;
```

However, things get more complicated when the two signals don't have the same time scale. A new time axis needs to be created that is the union of the two time axes. For instance, if signal $x_1[n]$ is defined for $-4 \leq n \leq 8$ and $x_2[n]$ is defined for $0 \leq n \leq 12$, then the sum $y[n] = x_1[n] + x_2[n]$ must be defined over $-4 \leq n \leq 12$.

Adding Signals in Matlab

The following program can be used to add two signals with different time axes:

```
function [y,n] = sigadd(x1,n1,x2,n2)
% implements y(n) = x1(n)+x2(n)
% -----------------------------
% [y,n] = sigadd(x1,n1,x2,n2)
%   y = sum sequence over n, which includes n1 and n2
%  x1 = first sequence over n1
%  x2 = second sequence over n2 (n2 can be different from n1)
%
n = min(min(n1),min(n2)):max(max(n1),max(n2)); % duration of y(n)
y1 = zeros(1,length(n));
y2 = y1;              % initialization
y1(find((n>=min(n1))&(n<=max(n1))==1))=x1;     % x1 with duration of y
y2(find((n>=min(n2))&(n<=max(n2))==1))=x2;     % x2 with duration of y
y = y1+y2;                                     % sequence addition
```

Exercise 1.17: Using "sigadd", add $x_1[n] = \exp(-n/4), -4 \leq n \leq 8$ with $x_2[n] = \cos(\pi n/4), 0 \leq n \leq 12$. Plot $x_1[n]$, $x_2[n]$, and $y[n]$.

1.5.2 Linear Combination of Signals

Suppose we have M signals $x_0[n], x_1[n], ..., x_{M-1}[n]$. We could add them all together to form a new signal:

$$y[n] = \sum_{k=0}^{M-1} x_k[n]. \tag{1.27}$$

More generally, we could multiply each signal $x_k[n]$ by a scalar value c_k to obtain a signal

$$y[n] = \sum_{k=0}^{M-1} c_k x_k[n]. \tag{1.28}$$

Such a signal is said to be a *linear combination* of the signals $x_0[n], x_1[n], ..., x_{M-1}[n]$.

Exercise 1.18: A moving average is a linear combination of the signals $x[n], x[n-1], ..., x[n-(M-1)]$. What are the values of the coefficients c_k?

Unit-Sample Synthesis

Any signal $x[n]$ can be represented as a linear combination of delayed unit-sample sequences:

$$x[n] = \sum_{k=n_1}^{n_2} x[k]\delta[n-k]. \tag{1.29}$$

Exercise 1.19: Represent the following sequence as a linear combination of delayed unit-sample sequences

$$x[n] = \left\{ 2, -\underset{\uparrow}{4}, 1, -2 \right\}. \tag{1.30}$$

Unit-Step Synthesis

Similarly, it is possible to represent $x[n]$ as a linear combination of unit-step sequences.

Exercise 1.20: Represent the previous $x[n]$ as a linear combination of delayed unit-step sequences.

1.5.3 Multiplication

Multiplication of two signals is analogous to addition, i.e. samples at the same time instant are added together. This is written as:

$$y[n] = x_1[n] \cdot x_2[n] \tag{1.31}$$

To multiply signals in Matlab with the same time scale, use '.*'

```
n = 0:12;
x1 = exp(-n/4);
x2= cos(pi*n/4);
y = x1.*x2;
```

The period before the asterisk is important because it signals Matlab to perform an element-by-element operation. Without the period, it will try to do a vector-vector multiply (which will fail due to mismatched dimensions).

To multiply signals with different axes, you may use:

```
function [y,n] = sigmult(x1,n1,x2,n2)
% implements y(n) = x1(n)*x2(n)
% -----------------------------
% [y,n] = sigmult(x1,n1,x2,n2)
%   y = product sequence over n, which includes n1 and n2
%  x1 = first sequence over n1
%  x2 = second sequence over n2 (n2 can be different from n1)
%
n = min(min(n1),min(n2)):max(max(n1),max(n2)); % duration of y(n)
y1 = zeros(1,length(n)); y2 = y1;              %
y1(find((n>=min(n1))&(n<=max(n1))==1))=x1;     % x1 with duration of y
y2(find((n>=min(n2))&(n<=max(n2))==1))=x2;     % x2 with duration of y
y = y1.* y2;
```

Exercise 1.21: Using "sigmult", multiply $x_1[n] = \exp(-n/4), -4 \leq n \leq 8$ with $x_2[n] = \cos(\pi n/4), 0 \leq n \leq 12$.

1.6 Problems

1. Consider the following sequences:

$$\begin{aligned} x[n] &= \{-4, 5, 1, -2, -3, 0, 2\}, -3 \le n \le 3 \\ y[n] &= \{6, -3, -1, 0, 8, 7, -2\}, -1 \le n \le 5 \\ w[n] &= \{3, 2, 2, -1, 0, -2, 5\}, 2 \le n \le 8. \end{aligned}$$

   The sample values of each of the above sequences outside the ranges specified are all zeros. Generate the following sequences:

   (a) $c[n] = x[-n+2]$.

   (b) $d[n] = y[-n-3]$.

   (c) $e[n] = w[-n]$.

   (d) $f[n] = x[n] + y[n-2]$.

   (e) $v[n] = x[n] \cdot w[n+4]$.

   (f) $s[n] = y[n] - w[n+4]$.

   (g) $r[n] = 3.5y[n]$.

   You may either do these by hand (i.e. with paper-and-pencil) or by using Matlab (in which case, you should turn in your plots and the commands used to generate them).

2. (a) Express the sequences $x[n], y[n]$ and $w[n]$ of Problem 1 as a linear combination of delayed unit sample sequences.

   (b) Express the sequences $x[n], y[n]$ and $w[n]$ of Problem 1 as a linear combination of delayed unit step sequences.

3. Compute the energy of each of the sequences $x[n], y[n]$ and $w[n]$ of Problem 1.

4. Plot each of the following sequences (using a stem diagram):

$$\begin{aligned} x_1[n] &= 3\delta[n+2] + 2\delta[n] - \delta[n-3] + 5\delta[n-7]. \\ x_2[n] &= \sum_{k=-5}^{5} e^{|k|}\delta[n-2k]. \\ x_3[n] &= 10u[n] - 5u[n-5] - 10u[n-10] + 5u[n-15]. \\ x_4[n] &= e^{0.1n}\left(u[n+20] - u[n-10]\right). \end{aligned}$$

   The sample values of each of the above sequences outside the ranges specified are all zeros. *Again, you may either do these by hand (i.e. with paper-and-pencil) or by using Matlab (in which case, you should turn in your plots and the commands used to generate them).*

Chapter 2

Discrete-Time Systems

2.1 Discrete-time Systems

A *discrete-time system* is an entity that processes one or more discrete-time *input sequence(s)* to produce a discrete-time *output sequence*. We have already seen many examples of *single-input* discrete-time systems:

1. **Time Shifter:** $y[n] = x[n - n_0]$, for a fixed delay n_0.
2. **Signal Folder:** $y[n] = x[-n]$.
3. **Scalar Multiplier:** $y[n] = cx[n]$, for a fixed multiplier c.
4. **Exponentiation:** $y[n] = (x[n])^k$, for a fixed exponent k.
5. **Accumulator:** $y[n] = x[n] + y[n - 1]$.
6. **FIR filter:** $y[n] = \sum_{k=0}^{M} b_k x[n - k]$, for a fixed set of coefficients $\{b_k\}$.
7. **Moving-average:** $y[n] = \sum_{k=0}^{M-1} \frac{1}{M} x[n - k]$, for a fixed value of M.

Discrete-time systems may also have two (or more) inputs. Examples of systems that have two inputs include:

1. **Adder:** $y[n] = x_1[n] + x_2[n]$.
2. **Mixer (a.k.a. Multiplier):** $y[n] = x_1[n] \cdot x_2[n]$.

Although multiple-input systems certainly have some important applications (such as signal cross-correlation), let's begin by discussing single-input systems in detail. Before delving into systems in general, let's look at one more specific system ...

2.2 Exponentially-Weighted Running Average Filter

Recall that the main problem with the accumulator is that it is *unstable.* This is evident because the step-response is a ramp function, and thus as $n \to \infty$, then $y[n] \to \infty$, i.e. the output tends to an infinite value, which will eventually exceed the maximum value that can be represented on the digital system. We can stabilize the filter by *dampening* the signal which is fed back to the input as follows:

$$y[n] \quad = \quad x[n] + \alpha y[n-1] \tag{2.32}$$

where $|\alpha| < 1$ is required to assure stability. An *exponentially-weighted running average filter* is a system that implements this equation.

2.2.1 Matlab Implementation

Exercise 2.1: Create a function called *expavg* which implements an exponentially-weighted running average filter. Use the *accumulate* function as a starting point. You will need an additional input, which is the value of α. The function should begin like this:

```
function [y,m] = expavg(x,n,alpha)
% implements y(n) = x(n) + alpha*y(n-1)
% ------------------------
% [y,m] = expavg(x,n,alpha)
%
```

Using your function, determine the *impulse* and *step* responses for $\alpha = 0.85$ over $-5 \leq n \leq 25$.

2.2.2 A Digital-Delay Effect

Digital Delay is a common effect used to process audio signals. A digital delay is merely an exponentially-weighted running average filter with a *delay* that may be more than just one sample. We can write this as:

$$y[n] \quad = \quad x[n] + \alpha y[n - n_0], \tag{2.33}$$

where the value of n_0 depends on the desired time delay. If the signal was sampled at a rate of F_T samples per second, then the delay in seconds T_d will be:

$$T_d \quad = \quad \frac{n_0}{F_T}. \tag{2.34}$$

Exercise 2.2: Create a function called *digitaldelay* which implements a digital delay effect. Use the *expavg* function as a starting point. You will need an additional input, which is the value of n_0. The function should look like this:

```
function [y,m] = digitaldelay(x,n,alpha,n0)
% implements y(n) = x(n) + alpha*y(n-n0)
% ------------------------
% [y,m] = digitaldelay(x,n,alpha,n0)
%
m = n; % no change in time axis
y = zeros(size(n)); % preallocate the output
y(1:n0) = x(1:n0); % first n0 output samples
start = n0+1; % start loop *after* the n0'th sample
for index=start:length(n)
   y(index) = x(index) + alpha*y(index-n0);
end
```

Let's use this function to process an actual audio signal stored in a *.wav* file (there is one on the eCampus page called *dave.wav* for you to use [1]). You will need to use the *wavread* function to read the wav file into Matlab and the *sound* function to play it through your speakers. I suggest that you append some zeros to the end of the sequence before running it through the digital delay (otherwise the output will get truncated to the length of the input, and you won't hear the trailing echoes).

```
[z,fs] = wavread( 'dave' ); % read the file
sound( z, fs ); % listen to the original signal
x = z'; % make the samples a row vector.
x = [x zeros(1,2*fs)]; % add two seconds of silence
n = 0:( length(x) - 1 ); % the discrete-time axis
alpha = 0.5; % dampening factor
Td = 0.5; % half-second delay
n0 = round(fs*Td); % integer number of samples in a half second
[y,m] = digitaldelay(x,n,alpha,n0); % pass through the digital delay
sound( y, fs ); % listen to the processed signal
```

You could repeat this for other *.wav* files that you can find on the Internet, and you could experiment with different delays T_d and dampening coefficients α. You can record your own audio by using the *wavrecord* function (Windows only) or by creating an *audiorecorder* object.

[1] Trivial: What movie is the clip from?

2.3 System Classifications

Systems can be classified in several different ways. They can be *linear* or *nonlinear*; they can be *time-invariant* or *time-varying*; they can be *stable* or *unstable*; they can be *causal* or *anti-causal*. Below, we will describe these four binary alternatives. In doing so, we will use the following notation:

$$y[n] \quad = \quad T\left(x[n]\right) \tag{2.35}$$

To represent the I/O relationship of a system.

2.3.1 Causality

A system is *causal* if the output $y[n_0]$ at time n_0 does not depend on future inputs $x[n], n > n_0$. A system that is not causal is said to be *anit-causal* or *anticipatory*. While strictly speaking, all physically-realizable systems must be causal, a system may exhibit anti-causal behavior if it processes recorded or delayed data (since when working with a recording, the processor has access to samples that are relatively earlier in time).

- **Exercise 2.3:** A particular FIR system is described by the I/O relationship

$$y[n] \quad = \quad \sum_{k=m_1}^{m_2} b_k x[n-k] \tag{2.36}$$

  for some integers m_1 and m_2, where $m_2 > m_1$. Is this FIR system necessarily causal? What must be true about m_1 to assure causality?

- **Exercise 2.4:** Is the signal folding operation causal?

2.3.2 Stability

A system is *stable* if for every *bounded* input $x[n]$, i.e. an input which satisfies,

$$|x[n]| \quad < \quad B_x, \qquad \text{for all } n$$

the corresponding output is *bounded*, i.e. it satisfies,

$$|y[n]| \quad < \quad B_y, \qquad \text{for all } n.$$

In the above two equations B_x and B_y represent finite constants.

- **Exercise 2.5:** Is the exponentially-weighted running average filter ($y[n] = x[n] + \alpha y[n-1]$) causal? What must be true of α to assure stability?

- **Exercise 2.6:** Is the FIR system ($y[n] = \sum_{k=0}^{M} b_k x[n-k]$) stable? You may assume all coefficients b_k are finite and that M is finite.

2.3.3 Linearity

Suppose we have a discrete-time system with I/O relationship $y[n] = T(x[n])$. It follows that:

$$\begin{aligned} y_1[n] &= T(x_1[n]) \\ y_2[n] &= T(x_2[n]), \end{aligned}$$

i.e., the output is $y_1[n]$ when the input is $x_1[n]$, and the output is $y_2[n]$ when the input is $x_2[n]$. Now create an input $x_c[n]$ which is a *linear combination* of $x_1[n]$ and $x_2[n]$,

$$x[n] = \alpha x_1[n] + \beta x_2[n]$$

where α and β are arbitrary coefficients. Then the system is said to be *linear* if

$$\begin{aligned} T(x[n]) &= T(\alpha x_1[n] + \beta x_2[n]) \\ &= \alpha T(x_1[n]) + \beta T(x_2[n]) \\ &= \alpha y_1[n] + \beta y_2[n] \\ &= \alpha T(x_1[n]) + \beta T(x_2[n]) \end{aligned} \tag{2.37}$$

A system that is not linear is said to be *non-linear.*

- **Exercise 2.7:** Is exponentiation $y[n] = (x[n])^k$ linear for $k \neq 1$?

- **Exercise 2.8:** Is the FIR system $y[n] = \sum_{k=0}^{M} b_k x[n-k]$ linear?

- **Exercise 2.9:** Is the system $y[n] = x[n] + c$ linear for $c \neq 0$?

- **Exercise 2.10:** Is the accumulator $y[n] = x[n] + y[n-1]$ linear?

Importance of Linearity

Recall that any signal may be expressed as a linear combination of delayed impulses (c.f. unit-sample synthesis),

$$x[n] \quad = \quad \sum_{k=-\infty}^{\infty} x[k]\delta[n-k]. \tag{2.38}$$

Let the above $x[n]$ be an input to a linear system. The output is

$$\begin{aligned} y[n] &= T\left(x[n]\right) \\ &= T\left(\sum_{k=-\infty}^{\infty} x[k]\delta[n-k]\right) \\ &= \sum_{k=-\infty}^{\infty} x[k]\underbrace{T\left(\delta[n-k]\right)}_{h[n,k]} \\ &= \sum_{k=-\infty}^{\infty} x[k]h[n,k] \end{aligned} \tag{2.39}$$

where $h[n,k]$ the response of the system to an impulse applied at time k.

2.3.4 Time-Invariance

A system is *time-invariant* or *shift-invariant* if a shift in the input signal produces a corresponding shift in the output. More precisely, if

$$T\left(x[n]\right) \quad = \quad y[n] \tag{2.40}$$

then

$$T\left(x[n-1]\right) \quad = \quad y[n-1]. \tag{2.41}$$

- **Exercise 2.11:** Is the system $y[n] = nx[n]$ time-invariant?

- **Exercise 2.12:** Is the folding operation time-invariant?

- **Exercise 2.13:** Is the FIR system ($y[n] = \sum_{k=0}^{M} b_k x[n-k]$) time-invariant?

2.4 LTI Systems and Convolution

We already saw that if the system is *linear*, then the output can be expressed as:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n,k] \tag{2.42}$$

where $h[n,k]$ is the response of an impulse applied at time k. Suppose an impulse is applied at time $k = 0$, i.e. the input is $x_0[n] = \delta[0]$, then the output is:

$$y_0[n] = h[n,0]. \tag{2.43}$$

Now, suppose that the impulse is delayed by one sample period so that the input is $x_1[n] = x_0[n-1] = \delta[1]$. The output is

$$y_1[n] = h[n,1]. \tag{2.44}$$

However, since it is also *time-invariant*, then it follows that $y_1[n] = y_0[n-1]$, and thus

$$y_1[n] = h[n-1,0]. \tag{2.45}$$

By equating equations (2.44) and (2.45), we see that

$$h[n,1] = h[n-1,0]. \tag{2.46}$$

It follows that

$$h[n,k] = h[n-k,0]. \tag{2.47}$$

That is, the response to an impulse applied at time k is simply a shifted version of the response to an impulse applied at time 0. Substituting (2.47) into (2.42), we get:

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k,0]. \tag{2.48}$$

Since we are now only concerned with the response to an impulse applied at time 0, we may drop the second argument of the function h and simply write it as $h[n-k]$

$$y[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]. \tag{2.49}$$

The function $h[n]$ is called the *impulse response* of the system and the above operation is called *convolution*,

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]. \tag{2.50}$$

Exercise 2.14: Convolve the following two sequences:

$$\begin{aligned} x[n] &= \{\underset{\uparrow}{5}, 1, 3\} \\ h[n] &= \{\underset{\uparrow}{6}, 4, 2\} \end{aligned}$$

Keeping Track of Time

A common mistake when convolving two sequences is to lose track of the time reference. The following rule should (hopefully) help you to avoid such a mistake:

- Suppose that $x[n]$ is nonzero over the range $N_{1,x} \leq n \leq N_{2,x}$ and $h[n]$ is nonzero over the range $N_{1,h} \leq n \leq N_{2,h}$.
- Then $y[n] = x[n] * h[n]$ will be nonzero over the range $(N_{1,x} + N_{1,h}) \leq n \leq (N_{2,x} + N_{2,h})$.
- Alternatively, you could say that $y[n]$ is nonzero over the range $N_{1,y} \leq n \leq N_{2,y}$ where $N_{1,y} = N_{1,x} + N_{1,h}$ and $N_{2,y} = N_{2,x} + N_{2,h}$.

The last interpretation provides guidance on how to implement convolution in Matlab.

2.4.1 Convolving in Matlab

The Matlab *conv* function will give the result of convolving two sequences. However, *conv* does not keep track of the time reference. In order to keep track of time, the following function, called *conv_m* should be used:

```
function [y,ny] = conv_m(x,nx,h,nh)
% Modified convolution routine for signal processing
% ----------------------------------------------------
% [y,ny] = conv_m(x,nx,h,nh)
%  y = convolution result
% ny = support of y
%  x = first signal on support nx
% nx = support of x
%  h = second signal on support nh
% nh = support of h
%
nyb = nx(1)+nh(1); nye = nx(length(x)) + nh(length(h));
ny = [nyb:nye];
y = conv(x,h);
```

Exercise 2.15: Repeat Exercise 2.14 using Matlab.

Causal LTI Systems

An LTI system with impulse response $h[n]$ is *causal* if and only if $h[n] = 0$ for all $n < 0$.

Stable LTI Systems

An LTI system with impulse response $h[n]$ is *stable* if and only if its impulse response is *absolutely summable*:

$$\sum_{n=-\infty}^{\infty} |h[n]| \quad < \quad \infty \tag{2.51}$$

Exercise 2.16: Is the exponentially-weighted running average: $y[n] = x[n] + \alpha y[n-1]$ stable?

2.5 Signal Correlation

There are many engineering applications where you need to compare the similarity of two signals:

1. **Ranging:** A reference radar or sonar signal is transmitted and reflected off a target. The received signal should be similar to the transmitted signal, except that it will contain random noise and will be delayed by a time that depends on the distance. The goal of the receiver is to estimate the delay, which will allow the range to be determined.

2. **Digital Communications:** Depending on the value of a data bit, one of two waveforms are transmitted over a noisy channel. The receiver needs to decide which waveform was transmitted by comparing the received signal against the two possible waveforms. Once the receiver determines which waveform was transmitted, it can determine the value of the data bit.

2.5.1 Cross-correlation

Correlation is a test of similarity between two signals. In its simplest form, the correlation of two signals $x[n]$ and $w[n]$ is merely

$$r_{xw} = \sum_{k=-\infty}^{\infty} x[n]w[n] \tag{2.52}$$

which is the sample sum of the product of the two signals. Suppose that $x[n]$ and $w[n]$ have the same energy $\mathcal{E}$ and that they may or may not be the same signal. If the two signals are the same ($x[n] = w[n]$), then r_{xw} will take on a large value, namely the energy of the signal, i.e. $r_{xw} = \mathcal{E}$. On the other hand, if the two signals are not the same ($x[n] \neq w[n]$) then r_{xw} will have a smaller magnitude, i.e. $|r_{xw}| \leq \mathcal{E}$.

Although simple, (2.52) suffers from an inability to tell if $w[n]$ is just a time-shifted version of $x[n]$. To handle time-shifts, we need a more generalized version of (2.52), which is called the *cross-correlation sequence*:

$$r_{xw}[\ell] = \sum_{n=-\infty}^{\infty} x[n]w[n-\ell] \tag{2.53}$$

where the parameter ℓ is called the *lag* and is an integer value.

2.5.2 Determining Time-Delay

Suppose that $x[n] = w[n - n_0]$, where n_0 is an unknown time offset. Then the value of the cross-correlation function will be at its maximum value at lag $\ell = n_0$

$$r_{xw}[n_0] = \sum_{n=-\infty}^{\infty} \underbrace{w[n-n_0]}_{x[n]} w[n-n_0] = \sum_{k=-\infty}^{\infty} w[k]w[k] = \mathcal{E} \tag{2.54}$$

where the change of variables $k = n - n_0$ was used in going from the first summation to the second one. At all other values of lag, $|r_{xw}| \leq \mathcal{E}$. Thus, to identify the the value of the delay n_0, all you need to do is compute the cross-correlation between $x[n]$ and $w[n]$ and then pick the value of ℓ for which the cross-correlation is at its peak,

$$n_0 = \arg\max r_{xw}[\ell] \tag{2.55}$$

where the "$\arg\max$" operator returns the index of the maximum value in the sequence $r_{xw}[\ell]$.

Exercise 2.17: Let $w[n]$ and $x[n]$ be:

$$\begin{aligned} w[n] &= \{\underset{\uparrow}{2}, 1, 4\} \\ x[n] &= \{\underset{\uparrow}{0}, 0, 2, 1, 4\}. \end{aligned}$$

As you can see, $x[n]$ is a delayed version of $w[n]$. Compute the cross-correlation sequence for $0 \leq \ell \leq 4$. Use the cross-correlation sequence to estimate the value of the time delay n_0.

2.5.3 Relationship with Convolution

We may also compute the cross-correlation sequence as:

$$r_{xw}[n] = x[n] * h[n] \tag{2.56}$$

where $*$ is the convolution operation

$$x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k] \tag{2.57}$$

and $h[n]$ is

$$h[n] = w[-n]. \tag{2.58}$$

The discrete-time system implementing this convolution is called a *matched filter*.

Proof:

Exercise 2.18: Redo Exercise 2.17 in Matlab using a matched filter.

2.5.4 Extended Application Example: Sonar

In a sonar system, a time-domain waveform is transmitted:

$$w(t) \quad = \quad \cos(\omega_c t), \quad \text{for } 0 \leq t \leq W \tag{2.59}$$

where $\omega_c = 2\pi f_c$ and f_c is the transmitted tone (in Hz). A typical value might be $f_c = 500$ Hz. The signal travels at the speed of sound, is reflected by a surface and is received as

$$x(t) \quad = \quad w(t - t_d) + \eta(t) \tag{2.60}$$

where $v(t)$ is noise. The time delay t_d is related to the distance to the object and the speed of sound v by:

$$2d \quad = \quad v t_d. \tag{2.61}$$

The speed of sound through room-temperature air is 331.3 m/sec. The factor of 2 in the above equation accounts for the fact that t_d is the *round-trip* delay. Solving for d, we find that the distance is:

$$d \quad = \quad \frac{v t_d}{2}. \tag{2.62}$$

Since this is a DSP class, we will implement the receiver digitally. Let's sample $x(t)$ at a rate of $f_s = 10$ kHz. The sampled signal becomes:

$$x[n] \quad = \quad x(nT) = w[n - n_d] + \eta[n] \tag{2.63}$$

where $\eta[n]$ is sampled noise and n_d is the time delay in units of *samples* (instead of units of seconds), found as:

$$n_d \quad = \quad round(t_d f_s) \tag{2.64}$$

where the rounding is necessary to make n_d an integer.

The goal of the receiver is to estimate n_d, which it can then use to determine the distance d to the target. The procedure is for the receiver to compute the cross-correlation between $x[n]$ and $w[n]$ and find the peak. The estimate n_d will then be the peak location of the autocorrelation function $r_{xw}[n]$.

The system can be implemented with the following Matlab code:

```
f_s = 10000; % sample frequency
f_c = 500;   % transmitted tone
d = 50;      % actual distance
v = 331.3;   % speed of sound (m/s)
t_d = 2*d/v; % delay (in seconds)
n_d = round(t_d*f_s); % delay (in samples)
nw = [0:f_s/20];  % simulate a pulse of 1/20 sec duration
w = cos( 2*pi*f_c*nw/f_s );  % sampled transmitted tone
[x,nx] = sigshift(w,nw,n_d); % delay the reflected signal
x = x + randn(size(nx));     % add noise
[h,nh] =  sigfold(w,nw);     % impulse response of matched filter
[r,nr] = conv_m( x, nx, h, nh ); % pass through matched filter
stem( nr, r );      % view the cross-correlation sequence
n_d_est = nr( find( r == max(r)) );
```

Exercise 2.19: Using the results of running the above code, estimate the distance d (in meters).

2.6 Problems

1. For each of the following discrete-time systems, determine whether or not the system is (1) linear, (2) causal, (3) stable, and (4) shift-invariant (In the following table, α and β are both nonzero constants and the function $round(x)$ rounds the sample x to the nearest integer):

Part	System	Linear?	Causal?	Time-invariant?	Stable?
(a)	$y[n] = n^3 x[n]$				
(b)	$y[n] = (x[n])^5$				
(c)	$y[n] = \beta + \sum_{\ell=0}^{3} x[n-\ell]$				
(d)	$y[n] = \ln(2 + \lvert x[n]\rvert)$				
(e)	$y[n] = \alpha x[-n+2]$				
(f)	$y[n] = x[n-4]$				
(g)	$y[n] = x[n]u[n]$				
(h)	$y[n] = x[n] + nx[n+1]$				
(i)	$y[n] = x[n] + \frac{1}{2}x[n-2] - \frac{1}{3}x[n-3]x[2n]$				
(j)	$y[n] = \sum_{k=-\infty}^{n+5} 2x[k]$				
(k)	$y[n] = x[2n]$				
(l)	$y[n] = round(x[n])$				

Just indicate "Yes" or "No" for each system and property (no proof needed).

2. Consider the following sequences:

   (i) $x_1[n] = 3\delta[n-2] - 2\delta[n+1]$.
   (ii) $x_2[n] = 5\delta[n-3] + 2\delta[n+1]$.
   (iii) $h_1[n] = -\delta[n+2] + 4\delta[n] - 2\delta[n-1]$.
   (iv) $h_2[n] = 3\delta[n-4] + 1.5\delta[n-2] - \delta[n+1]$.

   Determine the following sequences obtained by a linear convolution of a pair of the above sequences:

   (a) $y_1[n] = x_1[n] * h_1[n]$.
   (b) $y_2[n] = x_2[n] * h_2[n]$.
   (c) $y_3[n] = x_1[n] * h_2[n]$.
   (d) $y_4[n] = x_2[n] * h_1[n]$.

   Do the calculation by hand, but you can use Matlab to confirm your answer.

3. Consider the following sequences:

   (i) $x[n] = \{-4, 5, 1, -2, -3, 0, 2\}, -3 \leq n \leq 3$.
   (ii) $y[n] = \{6, -3, -1, 0, 8, 7, -2\}, -1 \leq n \leq 5$.
   (iii) $w[n] = \{3, 2, 2, -1, 0, -2, 5\}, 2 \leq n \leq 8$.

   Determine the following sequences obtained by a linear convolution of a pair of the above sequences:

   (a) $u[n] = x[n] * y[n]$.
   (b) $v[n] = x[n] * w[n]$.
   (c) $g[n] = w[n] * y[n]$.

   Do the calculation by hand, but you can use Matlab to confirm your answer.

4. A communication system uses the following signal to transmit a data '0'

$$\begin{aligned} w_0[n] &= \cos(\pi n), 0 \leq n \leq 5 \\ &= \left\{ \underset{\uparrow}{1}, -1, 1, -1, 1, -1 \right\} \end{aligned}$$

and the following signal to transmit a data '1'

$$\begin{aligned} w_1[n] &= \cos\left(\frac{2}{3}\pi n\right), 0 \leq n \leq 5 \\ &= \left\{ \underset{\uparrow}{1}, -\frac{1}{2}, -\frac{1}{2}, 1, -\frac{1}{2}, -\frac{1}{2} \right\} \end{aligned}$$

Suppose that the following signal is received:

$$x[n] = \left\{ \underset{\uparrow}{1}, -\frac{3}{4}, \frac{1}{2}, \frac{3}{4}, -\frac{3}{4}, 0 \right\}$$

Given the received signal $x[n]$, which is more likely to have been transmitted, data '0' or data '1'? To answer this question, please use the following procedure:

(a) Compute the cross-correlation $r_{x,w_0}[n]$ between $x[n]$ and $w_0[n]$ by using a matched filter. In particular, compute $r_{x,w_0}[n] = x[n] * h_0[n]$ where

$$h_0[n] = w_0[-n] = \left\{ -1, 1, -1, 1, -1, \underset{\uparrow}{1} \right\}.$$

Do the calculation by hand, but you can use matlab to confirm your answer.

(b) In a similar manner, find the cross-correlation $r_{x,w_1}[n]$ between $x[n]$ and $w_1[n]$.

(c) Next, compare $r_{x,w_0}[n]$ with $r_{x,w_1}[n]$ to determine which signal is more likely. This can be done by comparing the two cross-correlation sequences and identifying which of the two has the largest maximum value. A more careful analysis tells us that it is sufficient to compare the two sequences at sample $n = 0$, i.e. the *middle* sample of the output. Thus, if $r_{x,w_0}[0] > r_{x,w_1}[0]$ you can conclude that data '0' is more likely; otherwise data '1' is more likely. State which is more likely: data '0' or data '1'.

Chapter 3

The Discrete-Time Fourier Transform

3.1 Frequency Response

In this workbook, we are primarily concerned with designing digital filters that produce a desired *frequency response*. For instance, we might want to design a system that attenuates a signal component at a certain frequency (e.g. a 60 Hz signal leaking from a power supply). But before we can design for a frequency response, we need to first discuss what frequency response actually is.

Let's constrain ourselves to LTI systems. Recall that for an LTI system, the output $y[n]$ is found by convolving the input $x[n]$ with the *impulse* response $h[n]$ as follows:

$$y[n] = x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]. \tag{3.65}$$

Let the input to the LTI system be

$$x[n] = e^{j\omega n} \tag{3.66}$$

where $j = \sqrt{-1}$ is the imaginary number and ω is some *angular frequency*. From Euler's formula, we can also write it as:

$$x[n] = \cos[\omega n] + j\sin[\omega n]. \tag{3.67}$$

Exercise 3.1: Determine the output $y[n]$ to an LTI system when the input is $x[n] = e^{j\omega n}$.

The quantity $H\left(e^{j\omega}\right)$ is called the *frequency response* of the system. Notice that it is a complex-valued function of the continuous variable ω. Because it is complex-valued, we may decompose into a magnitude and phase,

$$H\left(e^{j\omega}\right) = |H\left(e^{j\omega}\right)|e^{j\theta(\omega)} \tag{3.68}$$

where $|H\left(e^{j\omega}\right)|$ is the *magnitude response* and $\theta(\omega)$ is the *phase response*.

At first glance, the fact that both $x[n]$ and $H\left(e^{j\omega}\right)$ are complex suggests that it cannot be measured if the LTI system only accepts real-valued inputs. However, by using linearity and (3.68), we can come up with a way to measure $H\left(e^{j\omega}\right)$. **Draw diagram here.**

Exercise 3.2: Consider a system that performs a time shift (delay). It is characterized by the impulse response

$$h[n] = \delta[n-n_0] \tag{3.69}$$

Determine the frequency response of this system.

Exercise 3.3: Consider a moving-average filter, which is characterized by the impulse response

$$h[n] = \frac{1}{M}\sum_{k=0}^{M-1}\delta[n-k] \tag{3.70}$$

Determine the frequency response of this system.

3.2 The DTFT

The operation for computing frequency response can be generalized for *any* discrete-time sequence $x[n]$ as

$$X\left(e^{j\omega}\right) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}. \tag{3.71}$$

This function is the *discrete-time Fourier transform* (DTFT) of $x[n]$, which we may denote as:

$$X\left(e^{j\omega}\right) = \mathcal{F}\left\{x[n]\right\}. \tag{3.72}$$

The DTFT is a complex function of frequency, with *magnitude* $|X\left(e^{j\omega}\right)|$ and phase $\theta(\omega)$, related by:

$$X\left(e^{j\omega}\right) = |X\left(e^{j\omega}\right)|e^{j\theta(\omega)}. \tag{3.73}$$

The frequency response is simply the DTFT of the impulse response,

$$H\left(e^{j\omega}\right) = \mathcal{F}\left\{h[n]\right\}. \tag{3.74}$$

3.2.1 Properties of the DTFT

The DTFT has the following important properties:

1. **Periodicity.** The DTFT is a periodic function of ω with period 2π radians.

2. **Conjugate Symmetry.** The DTF exhibits a conjugate symmetry

$$X\left(e^{j\omega}\right) \quad = \quad X^*\left(e^{-j\omega}\right). \tag{3.75}$$

This implies that the magnitude is an *even* function of ω:

$$\left|X\left(e^{-j\omega}\right)\right| \quad = \quad \left|X\left(e^{j\omega}\right)\right| \tag{3.76}$$

and the phase is an *odd* function of ω:

$$\theta(-\omega) \quad = \quad -\theta(\omega). \tag{3.77}$$

3.2.2 DTFT of a Finite Sequence

Suppose that $x[n]$ is nonzero over the range $N_1 \leq n \leq N_2$. Then the DTFT is

$$X\left(e^{j\omega}\right) \quad = \quad \sum_{n=N_1}^{N_2} x[n]e^{-j\omega n}. \tag{3.78}$$

and it has a finite number of terms.

Exercise 3.4: Determine the DTFT of the sequence:

$$x[n] = \{-3, \underset{\uparrow}{2}, 5, 6, 4\}$$

3.2.3 Computing the DTFT in Matlab

Suppose we are given a finite $x[n]$ and we want to plot its DTFT $X\left(e^{j\omega}\right)$ over the range $\omega_0 \leq \omega \leq \omega_M$. One problem we have is that ω is a continuous variable, but Matlab can only handle a finite number of values of ω. The solution to this problem is to *sample* the frequency axis. Let ω_k be the k^{th} sample of ω, with ω_0 being the first frequency sample and ω_M the last frequency sample. The sampled frequency axis can be generated using:

```
omega0 = -2*pi;  % first frequency sample (-2*pi is just an example)
omegaM = +2*pi;  % last frequency sample (+2*pi is just an example)
k = 0:M;
omega = omega0 + (omegaM-omega0)*k/M;
```

We can often set $\omega_0 = 0$ and $\omega_M = \pi$ so that the k^{th} frequency sample is simply

$$\omega_k = \pi\left(\frac{k}{M}\right). \tag{3.79}$$

Now we wish to evaluate (6.171). This could be done using a *for* loop, but we can do better than that by using *vector* operations. The operation given by (6.171) can be represented by the following vector inner product operation:

$$X\left(e^{j\omega_k}\right) = \begin{bmatrix} x[N_1] & ... & x[N_2] \end{bmatrix} \begin{bmatrix} e^{-j\omega_k N_1} \\ \vdots \\ e^{-j\omega_k N_2} \end{bmatrix}. \tag{3.80}$$

The above expression gives the DTFT for one frequency sample, i.e. the one for $\omega = \omega_k$. If we wish to determine the DTFT for multiple frequency samples, we can replace the above vector-vector multiplication with the following vector-matrix multiplication:

$$\begin{bmatrix} X\left(e^{j\omega_0}\right) & ... & X\left(e^{j\omega_M}\right) \end{bmatrix} = \begin{bmatrix} x[N_1] & ... & x[N_2] \end{bmatrix} \begin{bmatrix} e^{-j\omega_0 N_1} & ... & e^{-j\omega_M N_1} \\ \vdots & & \vdots \\ e^{-j\omega_0 N_2} & ... & e^{-j\omega_M N_2} \end{bmatrix}. \tag{3.81}$$

The result of the operation will be a row vector of length $M+1$ containing the DTFT at each frequency sample.

To summarize the above discussion, we can place the discrete-time signal $x[n]$ into the row vector $\mathbf{x}$ and the frequency-sampled DTFT $X\left(e^{j\omega_k}\right), \omega_k = \{\omega_0, ...\omega_M\}$, into the row vector $\mathbf{X}$. The DTFT can then be found from the vector-matrix operation:

$$\mathbf{X} = \mathbf{x}\mathbf{V} \tag{3.82}$$

where the matrix $\mathbf{V}$ is the matrix whose $(\ell, k)^{th}$ entry is

$$V_{\ell,k} = e^{-j\omega_k n_\ell} \tag{3.83}$$

where n_ℓ is the ℓ^{th} time sample and ω_k is the k^{th} frequency sample. The matrix $\mathbf{V}$ can be efficiently computed using the vector outer product:

$$\mathbf{V} = \exp\left(-j\mathbf{n}^T\boldsymbol{\omega}\right) \tag{3.84}$$

where $\mathbf{n}$ is a vector containing the time samples, and $\boldsymbol{\omega}$ is a vector containing the frequency samples.

The above discussion suggests the following code for determining the DTFT (we assume that omega, n, and x have already been defined):

```
V = exp( -j*n'*omega );  % the V matrix
X = x*V; % the DTFT
```

Exercise 3.5: Package the above code into a Matlab function with calling syntax:

```
function [X] = dtft( x, n, omega )
% Computes the DTFT
% X = dtft( x, n, omega )
% where X = DTFT at each value of omega
%       x = samples of a discrete-time signal
%       n = time indices for the signal
%       omega = vector of frequency samples
```

Using *your* function, compute and plot the DTFT of the following signal over $-2\pi \le \omega \le 2\pi$:

$$x[n] = \{-3, \underset{\uparrow}{2}, 5, 6, 4\}.$$

3.3 The Inverse DTFT

We sometimes need to invert the DTFT, i.e. obtain the sequence $x[n]$ from $X\left(e^{j\omega}\right)$. The mathematical definition of the *inverse* DTFT is:

$$x[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X\left(e^{j\omega}\right) e^{j\omega n} d\omega \tag{3.85}$$

and this operation can be represented by the shorthand notation:

$$x[n] = \mathcal{F}^{-1}\left\{X\left(e^{j\omega}\right)\right\}. \tag{3.86}$$

The inverse DTFT is rarely, if ever, found using (3.100). As with other frequency-domain transforms, a common way to invert is by using a table of transform pairs, which may be found in most common DSP textbooks.

Now let's consider how to implement an inverse DTFT operation in Matlab. One approach is to try to implement (3.100) using a numerical integration, but this is not the preferred way to perform an inverse DTFT. The preferred way is to solve for the vector $\mathbf{x}$ in (6.171), which can be easily done by recalling some rather simple matrix theory.

Define $\mathbf{V}^{-1}$ to be the *right* inverse of $\mathbf{V}$, which satisfies

$$\mathbf{V}\mathbf{V}^{-1} = \mathbf{I} \tag{3.87}$$

where $\mathbf{I}$ is an identity matrix. Post-multiply both sides of (3.83) by $\mathbf{V}^{-1}$,

$$\begin{aligned} \mathbf{X}\mathbf{V}^{-1} &= \mathbf{x}\mathbf{V}\mathbf{V}^{-1} \\ \mathbf{X}\mathbf{V}^{-1} &= \mathbf{x}\mathbf{I} \\ \mathbf{X}\mathbf{V}^{-1} &= \mathbf{x} \end{aligned} \tag{3.88}$$

Thus, we can see that the time-domain signal vector $\mathbf{x}$ can be found using:

$$\mathbf{x} = \mathbf{X}\mathbf{V}^{-1}. \tag{3.89}$$

Unless $\mathbf{V}$ is square, $\mathbf{V}^{-1}$ is actually a *pseudo*-inverse rather than an actual inverse[1]. In Matlab, $\mathbf{V}^{-1}$ is found using the *pinv* function. Thus the inverse DTFT may be implemented with the following single line (we assume that the matrix $\mathbf{V}$ has already been defined).

```
x = X*pinv(V);
```

[1]The *inverse* of a matrix A is only defined if $\mathbf{A}$ is *square*. The inverse is then defined as $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$

Exercise 3.6: Recall that the DTFT of the moving-average filter is

$$H\left(e^{j\omega}\right) \quad = \quad \frac{1}{M} e^{-j\omega(M-1)/2} \left(\frac{\sin(\omega M/2)}{\sin(\omega/2)} \right).$$

The above function is not defined for those ω that are integer multiples of 2π. This is because of the $\sin(\omega/2)$ in the denominator. For those locations, take $\sin(0)/\sin(0) = 1$. For instance, if we are interested in the range $-\pi \leq \omega \leq \pi$, then:

$$H\left(e^{j\omega}\right) \quad = \quad \begin{cases} 1 & \text{for } \omega = 0 \\ \frac{1}{M} e^{-j\omega(M-1)/2} \left(\frac{\sin(\omega M/2)}{\sin(\omega/2)} \right) & \text{otherwise} \end{cases}$$

Let $M = 10$. Represent $H\left(e^{j\omega}\right)$ in Matlab using 1001 samples over the range $-\pi \leq \omega \leq \pi$. By inverting the frequency-sampled DTFT, determine the impulse response $h[n]$ for $-20 \leq n \leq 20$.

3.4 Convolving in the Frequency Domain

As with other frequency-domain transforms, convolution in the time domain corresponds to multiplication in the frequency domain. That is, if the sequences $x[n]$, $h[n]$, and $y[n]$ are related by

$$y[n] = x[n] * h[n], \tag{3.90}$$

then their transforms are related by

$$Y\left(e^{j\omega}\right) = X\left(e^{j\omega}\right) H\left(e^{j\omega}\right). \tag{3.91}$$

Proof: Take the DTFT of both sides of (3.91),

$$\mathcal{F}\left\{y[n]\right\} = \mathcal{F}\left\{x[n] * h[n]\right\}. \tag{3.92}$$

Recall the definition of convolution:

$$x[n] * h[n] = \sum_{k=-\infty}^{\infty} x[k]h[n-k]. \tag{3.93}$$

Substitute (3.94) into (3.93)

$$\mathcal{F}\left\{y[n]\right\} = \mathcal{F}\left\{\sum_{k=-\infty}^{\infty} x[k]h[n-k]\right\}. \tag{3.94}$$

Use the linearity of the DTFT operator,

$$\mathcal{F}\left\{y[n]\right\} = \sum_{k=-\infty}^{\infty} x[k]\mathcal{F}\left\{h[n-k]\right\}. \tag{3.95}$$

To complete the proof, use the definition of the frequency response (DTFT of $h[n]$),

$$H\left(e^{j\omega}\right) = \sum_{n=-\infty}^{\infty} h[n]e^{-j\omega n}, \tag{3.96}$$

and perform an appropriate change of variables.

So convolution in the time domain corresponds to multiplication in the frequency domain,

$$Y\left(e^{j\omega}\right) = X\left(e^{j\omega}\right) H\left(e^{j\omega}\right) \tag{3.97}$$

where

$$\begin{aligned} X\left(e^{j\omega}\right) &= \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} \\ H\left(e^{j\omega}\right) &= \sum_{n=-\infty}^{\infty} h[n]e^{-j\omega n}. \end{aligned} \tag{3.98}$$

The time-domain output $y[n]$ can be found by inverting $Y\left(e^{j\omega}\right)$,

$$y[n] = \mathcal{F}^{-1}\left\{Y\left(e^{j\omega}\right)\right\} = \frac{1}{2\pi}\int_{-\pi}^{\pi} Y\left(e^{j\omega}\right) e^{j\omega n} d\omega. \tag{3.99}$$

Thus, an alternative way to implement convolution is as follows:

1. Compute $X\left(e^{j\omega}\right)$ and $H\left(e^{j\omega}\right)$ using (3.99).
2. Compute $Y\left(e^{j\omega}\right)$ using (3.98).
3. Compute $y[n]$ by inverting $Y\left(e^{j\omega}\right)$.

The above operations can be performed in Matlab by using the previously developed vector-based approaches for finding the forward and inverse DTFT's.

Exercise 3.7: Let $x[n]$ be given by

$$x[n] = \{-3, \underset{\uparrow}{2}, 5, 6, 4\}$$

and let $h[n]$ be a moving-average filter with $M = 10$. Determine the output $y[n]$ of the filter for $-20 \le n \le 20$. Using Matlab, perform this operation in both the time domain and the frequency domain. Compare your answers.

3.5 A Simple Filter Design

Suppose that the input to a digital filter is:

$$x[n] = \underbrace{\cos(\omega_1 n)}_{x_1[n]} + \underbrace{\cos(\omega_2 n)}_{x_2[n]}. \tag{3.100}$$

We would like to completely attenuate signal $x_2[n]$ and pass just signal $x_1[n]$. This requires that the magnitude response of the system be:

$$|H\left(e^{j\omega}\right)| = \begin{cases} 1 & \text{for } \omega = \omega_1 \\ 0 & \text{for } \omega = \omega_2. \end{cases} \tag{3.101}$$

Such a system can be implemented with an FIR filter with impulse response:

$$h[n] = \{\underset{\uparrow}{\alpha_0}, \alpha_1, \alpha_0\}. \tag{3.102}$$

The values of α_0 and α_1 will depend on the values of ω_1 and ω_2. To determine how these are related, take the DTFT of $h[n]$,

$$\begin{aligned} H\left(e^{j\omega}\right) &= \sum_{n=-\infty}^{\infty} h[n]e^{-j\omega n}. \\ &= \alpha_0 + \alpha_1 e^{-j\omega} + \alpha_0 e^{-j2\omega} \\ &= \alpha_0\left[1 + e^{-j2\omega}\right] + \alpha_1 e^{-j\omega} \\ &= \alpha_0 e^{-j\omega}\underbrace{\left[e^{j\omega} + e^{-j\omega}\right]}_{2\cos\omega} + \alpha_1 e^{-j\omega} \\ &= 2\alpha_0 e^{-j\omega}\cos\omega + \alpha_1 e^{-j\omega} \\ &= \left[2\alpha_0\cos\omega + \alpha_1\right]e^{-j\omega}. \end{aligned} \tag{3.103}$$

The magnitude response is:

$$\begin{aligned} \left|H\left(e^{j\omega}\right)\right| &= \left|\left[2\alpha_0\cos\omega + \alpha_1\right]e^{-j\omega}\right| \\ &= \left|2\alpha_0\cos\omega + \alpha_1\right|. \end{aligned} \tag{3.104}$$

If we substitute this back into (3.102), we get:

$$|2\alpha_0\cos\omega + \alpha_1| = \begin{cases} 1 & \text{for } \omega = \omega_1 \\ 0 & \text{for } \omega = \omega_2. \end{cases} \tag{3.105}$$

We are left with two equations and two unknowns, for which we can easily solve for α_0 and α_1.

Exercise 3.8: A certain power chord is made up of two notes: "A4" (440 Hz) and "E5" (659 Hz). Design a filter that passes the "A" and attenuates the "E". Assume the signal is sampled at 2 kHz.

3.6 Difference Equations

3.6.1 FIR Systems

A *finite impulse response* (FIR) system is an LTI system that has the following input/output relationship:

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] \tag{3.106}$$

where $\{b_k\}$ are the *feedforward filter coefficients.* Such a system has impulse response

$$h[n] = \left\{ \underset{\uparrow}{b_0}, b_1, ..., b_M \right\}. \tag{3.107}$$

The *order* of this system is M. The output of such a system is found by convolving the input with the impulse response, i.e. $y[n] = x[n] * h[n]$. The frequency response of such a system is given by:

$$H\left(e^{j\omega}\right) = \sum_{k=0}^{M} b_k e^{-j\omega k}. \tag{3.108}$$

In Matlab, the frequency response may be found by using the *dtft* function developed in this workbook.

3.6.2 IIR Systems

The output of the system given in (3.107) depends only on the current and past M inputs. We can build a much more flexible system if we allow the outputs to depend on N past *outputs.* Such a system could be expressed as follows:

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] - \sum_{k=1}^{N} a_k y[n-k]. \tag{3.109}$$

where $\{a_k\}$ are the *feedback filter coefficients.* When expressed like this, a system will often have an *infinite impulse response*, and when it does, it is called an *IIR* system. We can come up with an alternative expression for (3.110) by moving the second summation to the left side,

$$y[n] + \sum_{k=1}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k]. \tag{3.110}$$

If we let $a_0 = 1$, then this can be written as:

$$\sum_{k=0}^{N} a_k y[n-k] = \sum_{k=0}^{M} b_k x[n-k]. \tag{3.111}$$

An expression like (3.112) is called a *difference equation.* Almost every *physically realizable* discrete-time LTI system can be expressed as a difference equation.

Example Difference Equations

Many systems that we have already encountered can be expressed as a difference equation. For each of the following systems, determine the feedforward coefficients $\{b_0, ..., b_M\}$ and the feedback coefficients $\{a_0, ..., a_N\}$.

- **Exercise 3.9:** Moving average,

$$y[n] = \frac{1}{M+1} \sum_{k=0}^{M} x[n-k].$$

- **Exercise 3.10:** Exponentially weighted running average filter,

$$y[n] = x[n] + \alpha y[n-1].$$

- **Exercise 3.11:** Digital-delay effect,

$$y[n] = x[n] + \alpha y[n-n_0].$$

3.6.3 The Matlab *filter* Function

Rather than writing our own function for computing the output of a system expressed as a difference equation, we can use Matlab's *filter* function. Here is an excerpt of what you get when you type *help filter*:

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

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

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

There is one key difference between the *conv* function and the *filter function*. Suppose you would like to convolve a sequence $x[n] = \{x[0], x[1], ..., x[N]\}$ with an impulse response $h[n] = \{h[0], h[1], ..., h[M]\}$. With the *conv* function, the output will be all samples from $y[0]$ to $y[N + M]$. However, with the filter function, the output is truncated to be the same length as the input. Thus, the output will only be samples $y[0]$ through $y[N]$. This problem can be alleviated by *zero padding* the input, i.e. appending some extra zeros at the end of the input to ensure that the output has the desired length.

Exercises: Use the *filter* function to obtain the following outputs:

- **Exercise 3.12:** The output of an FIR system with impulse response:

$$h[n] = \{\underset{\uparrow}{6}, 4, 2\}$$

when the input is:

$$x[n] = \{\underset{\uparrow}{5}, 1, 3\}$$

Make sure you show the *entire* output.

- **Exercise 3.13:** The *impulse* and *step* responses over $-5 \leq n \leq 25$ for an exponentially-weighted running average filter with $\alpha = 0.85$.

- **Exercise 3.14:** The output of a system described by a difference equation with coefficients

$$\begin{aligned} b[n] &= \{\underset{\uparrow}{0.03}, 0.13, 0.24, 0.24, 0.13, 0.03\} \\ a[n] &= \{\underset{\uparrow}{1}, -1.11, 1.67, -1.21, 0.69, -0.25\} \end{aligned}$$

when the input is

$$x[n] = \underbrace{\cos(\omega_1 n)}_{x_1[n]} + \underbrace{\cos(\omega_2 n)}_{x_2[n]},$$

with $\omega_1 = 1.38$ and $\omega_2 = 2.07$.

3.6.4 Frequency-Response of a Difference Equation

Recall that if the input to an LTI system is:

$$x[n] = e^{j\omega n} \tag{3.112}$$

then the output is

$$y[n] = e^{j\omega n} H\left(e^{j\omega}\right). \tag{3.113}$$

From the time-invariance of the system, it follows that if the input is:

$$x[n-k] = e^{j\omega(n-k)} \tag{3.114}$$

then the output is

$$y[n-k] = e^{j\omega(n-k)} H\left(e^{j\omega}\right). \tag{3.115}$$

To determine the frequency response of a system specified by a difference equation, replace every $x[n-k]$ in the DE with the right-hand side of (3.115) and every $y[n-k]$ in the DE with the right-hand side of (3.116), then solve for $H\left(e^{j\omega}\right)$. **Show the derivation here:**

The resulting expression is

$$H\left(e^{j\omega}\right) = \frac{\sum_{k=0}^{M} b_k e^{-j\omega k}}{\sum_{k=0}^{N} a_k e^{-j\omega k}}. \tag{3.116}$$

If we let

$$\begin{aligned} b[n] &= \{\underset{\uparrow}{b_0}, b_1, ..., b_M\} \\ a[n] &= \{\underset{\uparrow}{a_0}, a_1, ..., a_N\} \end{aligned} \tag{3.117}$$

then

$$H\left(e^{j\omega}\right) = \frac{\mathcal{F}\{b[n]\}}{\mathcal{F}\{a[n]\}}. \tag{3.118}$$

This suggests a way to find the frequency response in Matlab: Find the DTFT of $b[n]$ and the DTFT of $a[n]$ using the *dtft* function we previously developed, then take the ratio of the two transforms according to (3.119).

Exercise 3.15: Find the frequency response for the IIR system whose difference equation coefficients are given in Exercise 3.14. Plot the magnitude response in *decibels* (dB).

3.6.5 The Matlab *freqz* and *fvtool* Functions

When the filter coefficients are stored in vectors **b** and **a**, then matlab can be used to compute the frequency response using the *freqz* function:

```
>> help freqz
 FREQZ Digital filter frequency response.
    [H,W] = FREQZ(B,A,N) returns the N-point complex frequency response
    vector H and the N-point frequency vector W in radians/sample of
    the filter:
               jw               -jw              -jmw
        jw  B(e)    b(1) + b(2)e + .... + b(m+1)e
      H(e) = ---- = ------------------------------------
               jw               -jw              -jnw
            A(e)    a(1) + a(2)e + .... + a(n+1)e
    given numerator and denominator coefficients in vectors B and A. The
    frequency response is evaluated at N points equally spaced around the
    upper half of the unit circle. If N isn't specified, it defaults to
    512.
```

Better yet, the *fvtool* function will not only compute the frequency response, but it will plot it:

```
>> help fvtool
 FVTOOL Filter Visualization Tool (FVTool).
    FVTool is a Graphical User Interface (GUI) that allows you to analyze
    digital filters.

    FVTOOL(B,A) launches the Filter Visualization Tool and computes
    the Magnitude Response for the filter defined by numerator and denominator
    coefficients in vectors B and A.

    FVTOOL(B,A,B1,A1,...) will perform an analysis on multiple filters.
```

Exercise 3.16: Use *freqz* and *fvtool* to find the frequency response for the IIR system whose difference equation coefficients are given in Exercise 3.14.

3.6.6 Inverting the Frequency Response

While finding the response given a set of filter coefficients is an important operation, a more interesting *design* problem arises from reversing the operation: Given a frequency response, can you find the corresponding filter coefficients? In some cases, it could be as easy as looking at the frequency response expression and identifying the coefficients.

Exercise 3.17: Given the following frequency response, determine the filter coefficients:

$$H\left(e^{j\omega}\right) = \frac{0.03\left(1+e^{-j5\omega}\right)+0.13\left(e^{-j\omega}+e^{-j4\omega}\right)+0.24\left(e^{-j2\omega}+e^{-j3\omega}\right)}{1-1.11e^{-j\omega}+1.67e^{-j2\omega}-1.21e^{-j3\omega}+0.69e^{-j4\omega}-0.25e^{-j5\omega}}.$$

The *invfreqz* Function

We might not always be able to pick off the coefficients so easily. It is desirable to have an *algorithmic* approach to finding the filter coefficients that correspond to a particular frequency response. In many cases, the frequency response can't be matched exactly with just a limited set of filter coefficients. When this is the case, we would like to find the filter coefficients that match the frequency response as closely as possible.

A way to proceed is as follows:

1. Specify a *target* frequency response $H_t\left(e^{j\omega}\right)$. This is the response for which we want to find a matching set of coefficients.

2. Pick a set of coefficients **a** and **b**.

3. Compute the *actual* frequency response $H_a\left(e^{j\omega}\right)$ of the system that uses these coefficients (i.e. by computing (3.119) with the coefficients found in step 2).

4. Determine the *mean square error* (MSE) between the target frequency response $H_t\left(e^{j\omega}\right)$ and the actual frequency response $H_a\left(e^{j\omega}\right)$ obtained in step 3. The MSE is given by:

$$\frac{1}{\pi}\int_0^{\pi}\left|H_t\left(e^{j\omega}\right)-H_a\left(e^{j\omega}\right)\right|^2 d\omega \tag{3.119}$$

5. If the MSE is acceptable, halt; otherwise go back to set 2 and try out a new set of coefficients.

As an alternative to this iterative technique, the frequency responses could be sampled and matlab used to solve an appropriately defined set of linear equations. This is a nontrivial task, but guess what: Matlab had a function to do it for you called *invfreqz*.

```
>> help invfreqz
 INVFREQZ  Discrete filter least squares fit to frequency response data.
    [B,A] = INVFREQZ(H,W,NB,NA) gives real numerator and denominator
    coefficients B and A of orders NB and NA respectively, where
    H is the desired complex frequency response of the system at frequency
    points W, and W contains the normalized frequency values within the
    interval [0, Pi] (W is in units of radians/sample).
```

3.6.7 Approximating an Ideal Lowpass Filter

An ideal lowpass filter is one which has the following frequency response:

$$H\left(e^{j\omega}\right) = \begin{cases} 1 & \text{for } 0 \leq \omega < \omega_c \\ 0 & \text{for } \omega_c \leq \omega \leq \pi \end{cases} \tag{3.120}$$

Exercise 3.18: Use *invfreqz* to compute an ideal lowpass filter with a target frequency response $H_t\left(e^{j\omega}\right)$ given by (3.121) using a cutoff $\omega_c = \pi/2$ radians. First, set the orders of numerator and denominator polynomials to $M = N = 5$. Plot the actual response of the system $H_a\left(e^{j\omega}\right)$ and compare with the target frequency response $H_t\left(e^{j\omega}\right)$. Repeat for order $M = N = 10$.

In doing the previous exercise, you will notice that there is an *overshoot* that arises due to the algorithms inability to match the discontinuity at ω_c. This is called the *Gibbs* phenomena. It can be alleviated by imposing a more gradual transition from the passband to the stopband, as given by:

$$H\left(e^{j\omega}\right) = \begin{cases} 1 & \text{for } 0 \leq \omega < \omega_1 \\ \frac{\omega_2-\omega}{\omega_2-\omega_1} & \text{for } \omega_1 \leq \omega < \omega_2 \\ 0 & \text{for } \omega_2 \leq \omega \leq \pi \end{cases} \tag{3.121}$$

Exercise 3.19: Repeat the previous exercise with $\omega_1 = 0.45\pi$ and $\omega_2 = 0.55\pi$.

3.6.8 The *fdatool*

We can do better than using *invfreqz* by realizing that within the transition band, we don't care what the frequency response is (as long as it is between 0 and 1). Furthermore, we can set tolerances on the passband and stopband ripple (the passband ripple is how far the actual frequency response deviates from the target frequeny response in the passband; similarly for the stopband ripple).

Matlab makes designing advanced filters a breeze with the *fdatool*. With fdatool, you specify the transition band and the tolerances, and pick from among several different design algorithms. It will give you the difference equation coefficients, which you can export to the matlab workspace and use to filter your data. You can even download the filters you create using fdatool to a Xilinx FPGA, or have Matlab implement your filter in C, VHDL, or Verilog which you can run on your DSP chip.

3.7 Problems

1. Using the "dtft" function developed in this chapter, compute the DTFT $X\left(e^{j\omega}\right)$ of the following finite-duration sequences over $0 \leq \omega \leq \pi$ with a frequency spacing of π/M radians (i.e. $M = 1000$). For each DTFT, use the subplot command to plot the magnitude $|X\left(e^{j\omega}\right)|$ in one subplot and the phase $\angle X\left(e^{j\omega}\right)$ in another subplot.

   (a) $x[n] = \left\{ \underset{\uparrow}{4}, 3, 2, 1, -1, -2, -3, -4 \right\}$.

   (b) $x[n] = (0.6)^{|n|} \left(u[n+10] - u[n-11] \right)$.

   (c) $x[n] = n(0.9)^n \left(u[n] - u[n-21] \right)$.

   (d) $x[n] = \cos[0.5\pi n] \left(u[n] - u[n-51] \right)$.

   Include your Matlab code for generating the plots (no need to turn in the dtft function).

2. For the following two sequences,

$$\begin{aligned} x[n] &= \left\{ -4, 5, 1, \underset{\uparrow}{-2}, -3, 0, 2 \right\} \\ h[n] &= \left\{ 6, \underset{\uparrow}{-3}, -1, 0, 8, 7, -2 \right\} \end{aligned}$$

   Determine $y[n] = x[n] * h[n]$ using a frequency-domain approach. More specifically, first determine $X(e^{j\omega})$ and $H(e^{j\omega})$ by taking the DTFT of $x[n]$ and $h[n]$. Next find $Y(e^{j\omega}) = X(e^{j\omega})H(e^{j\omega})$. Finally, take the inverse DTFT of $Y(e^{j\omega})$ to obtain $y[n]$. Compare your result against that obtained with time domain convolution (you may refer to the solution to problem 3(a) from chapter 2). Do this both analytically and in Matlab. The Matlab version will require you to write a function for computing the inverse DTFT. Turn in the following:

   (a) Your analytical solution showing expressions for $X(e^{j\omega})$, $H(e^{j\omega})$, $Y(e^{j\omega})$, and $y[n]$ plus your observation regarding how your $y[n]$ compares to the $y[n]$ from problem 3(a) in chapter 2.

   (b) Your Matlab function for computing the inverse DTFT.

   (c) Plots of the magnitude and phase of $X(e^{j\omega})$, $H(e^{j\omega})$, $Y(e^{j\omega})$ and the Matlab code used to produce the plots.

   (d) A stem plot that compares $y[n]$ computed using the frequency-domain approach and $y[n]$ computed using the time-domain approach (the solution to problem 3(a) from chapter 2).

3. An FIR filter filter of length 3 is defined by a symmetric impulse response; that is, $h[0] = h[2]$. Let the input to this filter be a sum of two cosine sequences of angular frequencies 0.3 rad/samples and 0.6 rad/samples, respectively. Determine the impulse response coefficients so that the filter passes only the high-frequency component of the input.

4. An FIR filter of length 5 is defined by a symmetric impulse response; that is $h[n] = h[4-n], 0 \leq n \leq 4$. Let the input to this filter be a sum of three cosine sequences of angular frequencies: 0.2 rad/samples, 0.5 rad/samples, and 0.8 rad/samples, respectively. Determine the impulse response coefficients so that the filter passes only the midfrequency component of the input.

5. For an LTI system described by the difference equation:

$$\sum_{k=0}^{N} a_k y[n-k] \quad = \quad \sum_{k=0}^{M} b_k x[n-k]$$

The frequency response is given by:

$$H\left(e^{j\omega}\right) \quad = \quad \frac{\sum_{k=0}^{M} b_k e^{-j\omega k}}{\sum_{k=0}^{N} a_k e^{-j\omega k}}.$$

Write a Matlab function called "`freqresp`" to compute the above frequency response. The format of the function should be:

```
function [H] = freqresp( b, a, omega )
% Computes the frequency response of the system defined by the
% difference equation coefficients contained in the vectors b and a
% at every frequency in the sampled frequency vector "omega"
% Usage: [H] = freqresp( b, a, omega )
%         H = the frequency response computed at the frequencies in omega
%         b = the feedforward difference equation coefficients
%         a = the feedback difference equation coefficients
%         omega = the sampled frequency axis
```

Use this function to compute the frequency response $H\left(e^{j\omega}\right)$ of the following systems over $0 \leq \omega \leq \pi$. For each frequency response, use the `subplot` command to plot the dB magnitude response $20\log_{10}|H\left(e^{j\omega}\right)|$ in one subplot and the unwrapped phase response $\angle H\left(e^{j\omega}\right)$ in another subplot.

(a) $y[n] = \frac{1}{5}\sum_{k=0}^{4} x[n-k]$.

(b) $y[n] = x[n] - x[n-2] + 0.95y[n-1] - 0.9025y[n-2]$.

(c) $y[n] = x[n] - x[n-1] + x[n-2] + 0.95y[n-1] - 0.9025y[n-2]$.

(d) $y[n] = x[n] - 1.7678x[n-1] + 1.58625x[n-2] + 1.1314y[n-1] - 0.64y[n-2]$.

(e) $y[n] = x[n] - \sum_{k=1}^{5}(0.5)^k y[n-k]$.

Include all of your Matlab code with your plots. The Matlab code and plots must be your own work (turning in someone else's code and/or plots is an honor code violation).

6. The following discrete-time signal:

$$x[n] \quad = \quad \left\{ \underset{\uparrow}{2}, 0, 1 \right\}$$

is passed through a linear time-invariant (LTI) system described by the difference equation:

$$y[n] + \frac{1}{2} y[n-2] \quad = \quad x[n] - \frac{1}{4} x[n-2]$$

Answer each of the following questions:

   (a) Write out an expression for the frequency response of the system, i.e. $H(e^{j\omega})$.
   (b) Write out an expression for Discrete-Time Fourier Transform (DTFT) of the input $x[n]$.
   (c) Write out an expression for the DTFT of the output $y[n]$.
   (d) Determine the output sequence $y[n]$.

7. The following discrete-time signal:

$$x[n] \quad = \quad \left\{ \underset{\uparrow}{0}, 2, 0, 4 \right\}$$

is passed through a linear time-invariant (LTI) system described by the difference equation:

$$y[n] \quad = \quad b_0 x[n] + b_1 x[n-1] + b_2 x[n-2] - a_1 y[n-1] - a_2 y[n-2]$$

The output of the system is the discrete-time signal:

$$y[n] \quad = \quad \left\{ \underset{\uparrow}{0}, 0, 10, 8 \right\}$$

Answer each of the following questions:

   (a) Write out an expression for the Discrete-Time Fourier Transform (DTFT) of the input $x[n]$.
   (b) Write out an expression for the Discrete-Time Fourier Transform (DTFT) of the output $y[n]$.
   (c) Determine the numerical values of the difference equation coefficients $\{b_0, b_1, b_2, a_1, a_2\}$.

Chapter 4

The z-Transfom

4.1 Definition of the z-Transform

Consider the DTFT of the unit-step function $u[n]$,

$$X\left(e^{j\omega}\right) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n} = \sum_{n=0}^{\infty} e^{-j\omega n}. \tag{4.122}$$

This is a geometric series of the form:

$$\sum_{n=0}^{\infty} c^n = \begin{cases} \frac{1}{1-c} & \text{if } |c| < 1 \\ \infty & \text{otherwise (i.e. does not converge)} \end{cases} \tag{4.123}$$

where in our case, $c = e^{-j\omega}$. What is the corresponding $|c|$? Does the DTFT of $u[n]$ converge?

If the DTFT doesn't converge, then it doesn't exist and it looks like we are out of luck. However, we can improve the situation if we replaced our $c = e^{-j\omega}$ with a new variable z^{-1} that satisfies the requirement $|z^{-1}| < 1$. This is the main idea behind the z-transform.

The z-Transform Operator

The z-transform of the sequence $x[n]$ is defined as:

$$X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n} = \mathcal{Z}\left\{x[n]\right\} \tag{4.124}$$

where z is a complex variable. The z-transform is more general than the DTFT, and hence, more flexible. In fact

$$X(e^{j\omega}) = X(z)\big|_{z=e^{j\omega}} \tag{4.125}$$

That is, the DTFT is the z-transform with $z = e^{j\omega}$.

Region of Convergence

Every z-transform is associated with a *region of convergence* (ROC), which specifies the values of z for which the transform converges. If z is outside the ROC, then the transform will not converge.

4.2 Some Important z-Transforms

4.2.1 z-Transform of $u[n]$

Exercise 4.1: Find the z-transform of $u[n]$, being careful to specify the ROC.

4.2.2 z-Transform of $-u[-n-1]$

Exercise 4.2: Now find the z-transform of $-u[-n-1]$, including its ROC.

4.2.3 Some Useful z-Transform Pairs

Here are some z-transform pairs that are especially useful.

sequence		z-transform	ROC
$\delta[n]$	$\Longleftrightarrow$	1	all z
$b^n u[n]$	$\Longleftrightarrow$	$\dfrac{1}{1-bz^{-1}}$	$\lvert z\rvert > \lvert b\rvert$
$-b^n u[-n-1]$	$\Longleftrightarrow$	$\dfrac{1}{1-bz^{-1}}$	$\lvert z\rvert < \lvert b\rvert$
$nb^n u[n]$	$\Longleftrightarrow$	$\dfrac{bz^{-1}}{(1-bz^{-1})^2}$	$\lvert z\rvert > \lvert b\rvert$
$r^n \cos[\omega_0 n]u[n]$	$\Longleftrightarrow$	$\dfrac{1-(r\cos\omega_0)z^{-1}}{1-(2r\cos\omega_0)z^{-1}+r^2z^{-2}}$	$\lvert z\rvert > \lvert r\rvert$

4.2.4 The z-Transform of Finite Sequences

Suppose that $x[n]$ is nonzero only over the range $N_1 \leq n \leq N_2$ and zero elsewhere. The z-transform is then:

$$X(z) = \sum_{n=N_1}^{N_2} x[n]z^{-n} = x[N_1]z^{-N_1} + ... + x[N_2]z^{-N_2}. \quad (4.126)$$

The ROC will include the entire z-plane except that $z = 0$ and/or $z = \infty$ *might* not be in the ROC.

Exercise 4.3: Determine the z-transform of the folloiwing sequence:

$$x[n] = \{\underset{\uparrow}{1}, 3, 2\}.$$

Make sure to specify the ROC. For this transform, determine the values of z for which $X(z) = 0$. These are the *zeros* of the transform.

Exercise 4.4: Let $y[n] = x[-n]$, where $x[n]$ is as specified in Exercise 4.3. Determine the z-transform, including the ROC.

4.3 Computing the z-Transform with Matlab

Suppose we have a finite sequence $x[n]$, which we may represent in matlab by the vectors **x** and **n**. We would like to determine the z-transform. The key to doing this is to represent the complex variable z in polar form,

$$z = re^{j\omega} \tag{4.127}$$

Substituting this into the definition of the z-transform, we get

$$\begin{aligned} X(z) &= \sum_{n=-\infty}^{\infty} x[n]\left(re^{j\omega}\right)^{-n} \\ &= \sum_{n=-\infty}^{\infty} x[n]r^{-n}e^{-j\omega n} \\ &= \sum_{n=-\infty}^{\infty} \left(x[n]r^{-n}\right)e^{-j\omega n} \\ &= \mathcal{F}\left\{x[n]r^{-n}\right\} \end{aligned} \tag{4.128}$$

This equation shows us that the z-transform evaluated at radius r is the DTFT of the *modified* sequence $x[n]r^{-n}$.

This suggests a method for computing the z-transform in Matlab if the sequence is finite.

1. Create a sampled frequency axis *omega*.
2. Create a sampled radius axis *r*.
3. For each value of r, determine the DTFT of the sequence $x[n]r^{-n}$

The corresponding matlab code looks as follows:

```
NumFreq = 1000; % one less than the number of frequency samples
NumRad = 100; % number of radius samples
omega = 2*pi*[0:NumFreq]/NumFreq; % sampled frequency axis
MaxRad = 3; % maximum radius r
r = MaxRad*[1:NumRad]/NumRad; % the sampled radius axis
z = r'*exp(-j*omega); % the z-axis
for i=1:NumRad
  X(i,:) = dtft( x.*r(i).^-n, n, omega );  % DTFT of modified signal
end
mesh( real(z), imag(z), 20*log10( abs(X) ) ); %visualize in 3-D
view(0,0); % look at it from the side.
```

4.4 Useful z-transform Properties

The z-transform has properties that are similar to that of the DTFT, namely:

Linearity: $\mathcal{Z}\left\{ax[n] + by[n]\right\} = a\mathcal{Z}\left\{x[n]\right\} + b\mathcal{Z}\left\{y[n]\right\} = aX(z) + bY(z)$.

Sample Shift: $\mathcal{Z}\left\{x[n-n_0]\right\} = z^{-n_0}\mathcal{Z}\left\{x[n]\right\} = z^{-n_0}X(z)$.

Convolution: $\mathcal{Z}\left\{x[n] * y[n]\right\} = \mathcal{Z}\left\{x[n]\right\}\mathcal{Z}\left\{y[n]\right\} = X(z)Y(z)$.

4.5 Convolution Using the z-Transform

Just as we can implement convolution by multiplying DTFT's, we can also implement it by multiplying z-transforms.

Exercise 4.5: Using the z-transform, convolve the following two sequences:

$$
\begin{aligned}
x[n] &= \{\underset{\uparrow}{5}, 1, 3\} \\
h[n] &= \{\underset{\uparrow}{6}, 4, 2\}
\end{aligned}
$$

4.6 System Function

The *system function* $H(z)$ of a LTI system is the z-transform of its impulse response, i.e.

$$H(z) \quad = \quad \mathcal{Z}\{h[n]\}. \tag{4.129}$$

The frequency-response of the system is simply the system function evaluated at $z = e^{j\omega}$.

Exercise 4.6: Consider a discrete-time system represented by a difference equation:

$$\sum_{k=0}^{N} a_k y[n-k] \quad = \quad \sum_{k=0}^{M} b_k x[n-k]. \tag{4.130}$$

Determine an expression for the *system function* in terms of the coefficients $\{a_0, ...a_N\}$ and $\{b_0, ..., b_M\}$. To do this, take the z-transform of both sides of (4.131) and solve for $H(z) = Y(z)/X(z)$.

Exercise 4.7: Find the system function for the difference equation:

$$y[n] - \frac{3}{4}y[n-1] + \frac{1}{8}y[n-2] \quad = \quad x[n] + 3x[n-1] + 2x[n-2]. \tag{4.131}$$

4.6.1 Poles and Zeros

Let $H(z) = B(z)/A(z)$, then

- The *poles* of the system are the *roots*[1] of the denominator polynomial $A(z)$.
- The *zeros* of the system are the roots of the numerator polynomial $B(z)$.

A *pole-zero* plot is a plot showing the locations of the poles and zeros on the complex plane. By convention, poles are represented by '$\times$' and zeros are represented by '$\circ$'.

[1]A root of $A(z)$ is a value of z for which $A(z) = 0$.

Exercise 4.8: Determine the poles and zeros of the system specified by difference equation (4.132). Sketch this system's pole-zero plot. *Hint: You may use Matlab's* roots *function* .

4.6.2 ROC and Poles

If a system is *causal*, then its impulse response is a right-sided sequence, and the region of convergence of $H(z)$ is

$$|z| \quad > \quad \max |\lambda_k|$$

where λ_k are the poles.

Exercise 4.9: Determine the ROC for the system specified by difference equation (4.132).

4.6.3 Stability

Bounded-input, bounded-output (BIBO) stability requires that $H(e^{j\omega})$ exists. This can only be true if the ROC contains the unit circle. If the system is causal, then the ROC will be the region that lies outside the outermost pole. Therefore, all poles must be inside the unit circle for the system to be stable.

Exercise 4.10: Is the system with difference equation (4.132) stable?

4.6.4 Factored Form

Let $\{\xi_1, ..., \xi_M\}$ be the zeros of the system, and let $\{\lambda_1, ..., \lambda_N\}$ be the poles. The system function may be written as:

$$H(z) \quad = \quad \left(\frac{b_0}{a_0}\right) \frac{\prod_{k=1}^{M}(1-\xi_k z^{-1})}{\prod_{k=1}^{N}(1-\lambda_k z^{-1})} = \frac{b_0(1-\xi_1 z^{-1})(1-\xi_2 z^{-1})...(1-\xi_M z^{-1})}{a_0(1-\lambda_1 z^{-1})(1-\lambda_2 z^{-1})...(1-\lambda_N z^{-1})} \tag{4.132}$$

This is called *factored* form, and the quantity b_0/a_0 is called the *system gain.*

Exercise 4.11: Express the system function for the system with difference equation (4.132) in factored form.

What happens when $N \neq M$?

We know that that if $N > M$, then there will be $N - M$ extra zeros located at $z = 0$ and if $M > N$ there will be $M - N$ extra poles located at $z = 0$. This is required to make the number of poles and zeros the same. However, these extra poles or zeros are not apparent in equation (4.133), so where are they? To reveal them, you need to express the system function in terms of z instead of z^{-1}. When $N > M$, this is done by multiplying the numerator and denominator of (4.133) by z^N, which results in

$$H(z) \quad = \quad z^{N-M}\left(\frac{b_0}{a_0}\right) \frac{\prod_{k=1}^{M}(z-\xi_k)}{\prod_{k=1}^{N}(z-\lambda_k)}. \tag{4.133}$$

Similarly, if $M > N$, then multiply numerator and denominator of (4.133) by z^M, which results in the same expression as (4.134). The expression (4.134) is also a factored form of the system function.

Exercises

For each of the following difference equations, represent the system function in the factored form given by (4.134). Furthermore, Sketch a *pole-zero* plot, and determine if the system is stable.

- **Exercise 4.12:** $y[n] = x[n] - 1.2x[n-1] + x[n-2] + 1.3y[n-1] - 1.04y[n-2] + 0.222y[n-3]$.

- **Exercise 4.13:** $y[n] = x[n] - 1.3x[n-1] + 1.04x[n-2] - 0.222x[n-3] + 1.2y[n-1] - y[n-2]$.

4.6.5 Getting the Difference Equation from Poles and Zeros

If we are given the poles, zeros, and system gain b_0/a_0, we can immediately find the system function in the factored form (4.133). If we would like to get the difference equation coefficients, then it is a matter of converting the system function from the factored form into the *polynomial* form,

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{\sum_{k=0}^{N} a_k z^{-k}}. \tag{4.134}$$

This can be accomplished in matlab by using the *poly* function.

Exercise 4.14: A system has system gain $b_0/a_0 = 2$ and the following set of poles and zeros:

$$\begin{aligned} \xi_k &= \{+1, +1\} \\ \lambda_k &= \left\{+\frac{3}{4}, +\frac{1}{2}\right\} \end{aligned}$$

Write out a difference equation for the system.

4.7 The Inverse z-Transform

The z-transform of a finite sequence $x[n] = \{x[N_1], ..., x[N_2]\}$ is

$$H(z) = \sum_{k=N_1}^{N_2} x[k]z^{-k} \tag{4.135}$$

and is easily inverted:

$$h[n] = \mathcal{Z}^{-1}\left\{\sum_{k=N_1}^{N_2} x[k]z^{-k}\right\} = \sum_{k=N_1}^{N_2} x[k]\mathcal{Z}^{-1}\left\{z^{-k}\right\} = \sum_{k=N_1}^{N_2} x[k]\delta[n-k]. \tag{4.136}$$

But what if the sequence is not finite? How do you invert the z-transform? Oftentimes, it requires a *partial-fraction expansion.*

4.7.1 Partial-Fraction Expansion

When there are no repeated poles and the sequence is *causal*, we can usually get by using the following two z-transform pairs

sequence		z-transform	ROC
$b^n u[n]$	$\longleftrightarrow$	$\dfrac{1}{1-bz^{-1}}$	$\lvert z\rvert > \lvert b\rvert$
$\delta[n-k]$	$\longleftrightarrow$	z^{-k}	$\lvert z\rvert > 0$ if $k > 0$

along with the concept of *partial-fraction expansion* (PFE).

Exercise 4.15: Invert the following z-transform:

$$H(z) = \frac{1 - z^{-1}}{1 - \frac{3}{4}z^{-1} + \frac{1}{8}z^{-2}}.$$

Note that this is a *proper* fraction (since the degree of $B(z)$ is less than the degree of $A(z)$).

4.7.2 Dealing with Improper Fractions

Let $H(z) = B(z)/A(z)$ be in *rational form.* Recall that M is the degree of $B(z)$ and N is the degree of $A(z)$. If $M \geq N$, then $H(z)$ is an *improper* fraction. Before applying partial-fraction expansion, it needs to be converted into the sum of a polynomial and a proper fraction through polynomial *long division.*

$$H(z) \quad = \quad C(z) + \frac{\tilde{B}(z)}{A(z)} \tag{4.137}$$

where $C(z)$ is the quotient and $\tilde{B}(z)$ is the remainder when $B(z)$ is divided by $A(z)$, i.e.

$$B(z) \quad = \quad C(z)A(z) + \tilde{B}(z). \tag{4.138}$$

By performing a *partial-fraction expansion* on the proper fraction $\tilde{B}(z)/A(z)$, $H(z)$ may be expressed as:

$$H(z) \quad = \quad \underbrace{\sum_{k=0}^{M-N} C_k z^{-k}}_{C(z)} + \sum_{k=1}^{N} \frac{R_k}{1 - \lambda_k z^{-1}}. \tag{4.139}$$

The *direct-term* polynomial $C(z)$ is the quotient obtained by dividing $B(z)$ by $A(z)$, while R_k is the *residue* associated with pole λ_k. Once the z-transform is expressed as given in (4.140), the inverse z-transform is easily found as:

$$h[n] \quad = \quad \sum_{k=0}^{M-N} C_k \delta[n-k] + \sum_{k=1}^{N} R_k \left(\lambda_k\right)^n u[n]. \tag{4.140}$$

Exercise 4.16: Invert:

$$H(z) \quad = \quad \frac{5 - 4z^{-1} + \frac{1}{2}z^{-2}}{1 - \frac{3}{4}z^{-1} + \frac{1}{8}z^{-2}}.$$

4.7.3 Inverting the Z-Transform in Matlab

Matlab can do the partial-fraction expansion for you. Just use the *residuez* function:

```
>> help residuez
 RESIDUEZ Z-transform partial-fraction expansion.
    [R,P,K] = RESIDUEZ(B,A) finds the residues, poles and direct terms
    of the partial-fraction expansion of B(z)/A(z),

       B(z)       r(1)               r(n)
       ---- = ------------ +...  ------------ + k(1) + k(2)z^(-1) ...
       A(z)   1-p(1)z^(-1)       1-p(n)z^(-1)

    B and A are the numerator and denominator polynomial coefficients,
    respectively, in ascending powers of z^(-1).  R and P are column
    vectors containing the residues and poles, respectively.  K contains
    the direct terms in a row vector.  The number of poles is
       n = length(A)-1 = length(R) = length(P)
    The direct term coefficient vector is empty if length(B) < length(A);
    otherwise,
       length(K) = length(B)-length(A)+1

    If P(j) = ... = P(j+m-1) is a pole of multiplicity m, then the
    expansion includes terms of the form
            R(j)              R(j+1)                   R(j+m-1)
       -------------- + ------------------   + ... + ------------------
       1 - P(j)z^(-1)   (1 - P(j)z^(-1))^2           (1 - P(j)z^(-1))^m

    [B,A] = RESIDUEZ(R,P,K) converts the partial-fraction expansion back
    to B/A form.
```

Exercise 4.17: Repeat the last two examples in Matlab using the *residuez* function.

4.7.4 Repeated Poles

If a particular pole λ has *multiplicity* m, then its terms are expanded as:

$$H(z) \;=\; \sum_{\ell=1}^{m} \frac{R_\ell}{\left(1-\lambda z^{-1}\right)^\ell} \tag{4.141}$$

where

$$R_\ell \;=\; \frac{1}{(\ell-1)!} \lim_{z\to\lambda} \frac{d^{\ell-1}}{dz^{\ell-1}} \Big((z-\lambda)^\ell H(z) \Big). \tag{4.142}$$

To invert the first two terms of this summation, we can use the following z-transform pairs:

sequence		z-transform	ROC
$b^n u[n]$	$\longleftrightarrow$	$\dfrac{1}{1-bz^{-1}}$	$\lvert z\rvert > \lvert b\rvert$
$(1+n)b^n u[n]$	$\longleftrightarrow$	$\dfrac{1}{(1-bz^{-1})^2}$	$\lvert z\rvert > \lvert b\rvert$

If there are terms beyond this, they may be inverted by applying the *differentiation* property of the z-transform. To keep things manageable, we will limit multiplicities in this class to be no more than 2.

Exercise 4.18: Determine the impulse response of the system described by difference equation:

$$y[n] + \frac{3}{18}y[n-1] - \frac{2}{9}y[n-2] - \frac{1}{18}y[n-3] \;=\; x[n].$$

Exercise 4.19: Suppose that the input to the system given by Example 4.18 is:

$$x[n] \quad = \quad \left\{ \underset{\uparrow}{2}, 1, 3 \right\}.$$

Determine the output over the range $0 \leq n \leq 20$. Find the output two ways: (1) Using the *filter* function along with the difference-equation coefficients, and (2) Using the *conv* function along with the impulse response.

4.7.5 Complex Poles

If a pole λ is complex, then the inverse z-transform $\mathcal{Z}^{-1}\left\{1/(1-\lambda z^{-1})\right\} = \lambda^n u[n]$ will produce a complex-valued sequence. This is not the preferred representation of the output. Instead, we would like the output to be expressed as a real-valued sequence. Fortunately, complex poles always occur in *conjugate* pairs. Furthermore, the residues of a conjugate pair of poles are themselves a conjugate pair. For each conjugate pair, there will be a pair of terms in the PFE of the following form:

$$H(z) = \frac{R}{1-\lambda z^{-1}} + \frac{R^*}{1-\lambda^* z^{-1}} \tag{4.143}$$

which will have inverse z-transform:

$$h[n] = \left(R\left(\lambda\right)^n + R^*\left(\lambda^*\right)^n\right)u[n]. \tag{4.144}$$

Recall that complex-valued residues R and poles λ can be represented as:

$$\begin{aligned} R &= |R|e^{j\angle R} \\ \lambda &= |\lambda|e^{j\angle\lambda} \end{aligned} \tag{4.145}$$

where $|R|$ is the magnitude of R and $\angle R$ is the angle of R, and similarly $|\lambda|$ is the magnitude of λ and $\angle\lambda$ is the angle of λ. Thus we can express:

$$\begin{aligned} R\left(\lambda\right)^n + R^*\left(\lambda^*\right)^n &= |R|e^{j\angle R}\left(|\lambda|e^{j\angle\lambda}\right)^n + |R^*|e^{j\angle R^*}\left(|\lambda^*|e^{j\angle\lambda^*}\right)^n \\ &= |R||\lambda|^n\left(e^{j(\angle R+n\angle\lambda)} + e^{-j(\angle R+n\angle\lambda)}\right) \\ &= 2|R||\lambda|^n\cos\left(\angle R+n\angle\lambda\right) \end{aligned} \tag{4.146}$$

and it follows that the inverse z-transform for the conjugate pair of poles is:

$$h[n] = 2|R||\lambda|^n\cos\left(\angle R+n\angle\lambda\right)u[n]. \tag{4.147}$$

Exercise 4.20: Determine the impulse response of the system described by difference equation:

$$y[n] - 0.7y[n-1] + 0.44y[n-2] + 0.222y[n-3] = 4x[n] - 3.24x[n-1] + 0.928x[n-2].$$

Express the impulse response as a real-valued sequence.

4.8 The *poly* Function

Suppose that instead of a difference equation or (polynomial form) system function, we are given a set of poles and zeros along with the system gain b_0/a_0. We want to determine the impulse response. The "b" coefficients can be found from the zeros, and the "a" coefficients can be found from the poles. In matlab, the *poly* function will find the appropriate polynomial coefficients:

```
>> help poly
 POLY Convert roots to polynomial.
    POLY(A), when A is an N by N matrix, is a row vector with
    N+1 elements which are the coefficients of the
    characteristic polynomial, DET(lambda*EYE(SIZE(A)) - A) .

    POLY(V), when V is a vector, is a vector whose elements are
    the coefficients of the polynomial whose roots are the
    elements of V . For vectors, ROOTS and POLY are inverse
    functions of each other, up to ordering, scaling, and
    roundoff error.
```

If the system gain is not equal to one, then simply multiply all the "b" coefficients by the gain. Once the "a" and "b" coefficients are found, the system function can be inverted with the help of the *residuez* function.

Exercise 4.21: Suppose a system has zeros

$$\xi = \left\{ \frac{4}{5}, \frac{3}{5}, \frac{2}{5}, \frac{1}{5} \right\}$$

and poles

$$\lambda = \left\{ \frac{3}{4}, \frac{1}{2}, \frac{1}{4}, 0 \right\}$$

and system gain $b_0/a_0 = 2$. Find the corresponding impulse response.

4.9 Problems

For the problems in this chapter, you may use matlab's roots *function or your calculator to determine poles and zeros.*

1. A particular discrete-time system can be represented by the following difference-equation:

$$y[n] + \frac{1}{2}y[n-1] - \frac{3}{16}y[n-2] \quad = \quad x[n] + x[n-1] + \frac{1}{4}x[n-2]$$

   (a) Determine the system function, including the region of convergence.

   (b) Determine the poles and zeros of this system.

   (c) Is this system BIBO stable?

   (d) Put the system function into factored form:

$$H(z) \quad = \quad \left(\frac{b_0}{a_0}\right)\left(\frac{(1-\xi_1 z^{-1})(1-\xi_2 z^{-1})}{(1-\lambda_1 z^{-1})(1-\lambda_2 z^{-1})}\right)$$

2. Determine the transfer function of each of the following causal LTI discrete-time systems described by the difference equations. Express each transfer function in factored form and sketch its pole-zero plot. Is the corresponding system BIBO stable?

   (a) $y[n] = 5x[n]+9.5x[n-1]+1.4x[n-2]-24x[n-3]+0.1y[n-1]-0.14y[n-2]-0.49y[n-3]$.

   (b) $y[n] = 5x[n]+16.5x[n-1]+14.7x[n-2]-22.04x[n-3]-33.6x[n-4]+0.5y[n-1]-0.1y[n-2]$ $-0.3y[n-3] + 0.0936y[n-4]$.

3. A causal LTI system has impulse response:

$$h[n] \quad = \quad n\left(\frac{1}{3}\right)^n u[n] + \left(-\frac{1}{4}\right)^n u[n].$$

   For this system determine:

   - The system function $H(z)$, including the region of convergence.
   - The difference equation.
   - A list of poles and zeros along with the system gain b_0/a_0.

   In addition, determine if this system is BIBO stable.

4. A causal linear time-invariant (LTI) discrete-time system has a pair of poles and a pair of zeros. The poles are $\lambda_1 = \frac{1}{4} + j\frac{\sqrt{3}}{4}$ and $\lambda_2 = \frac{1}{4} - j\frac{\sqrt{3}}{4}$. The zeros are $\xi_1 = 0$ and $\xi_2 = \frac{1}{4}$. The system gain is $b_0/a_0 = 4$.

   (a) Is the system stable?

   (b) Write the difference equation for this system using only real-valued coefficients. You may assume that $a_0 = 1$.

   (c) The impulse response of this system may be written in the form:

   $$h[n] = c\,(r)^n \cos\left[\omega_0 n\right] u[n]$$

   Determine the values of the coefficients c, r and ω_0.

   (d) Suppose that the input to system is the sequence:

   $$x[n] = \left\{ \underset{\uparrow}{4}, -2, 1 \right\}$$

   Determine the corresponding output sequence.

Chapter 5

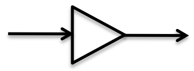
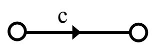
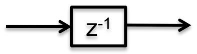
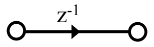
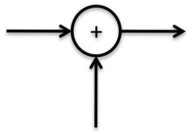
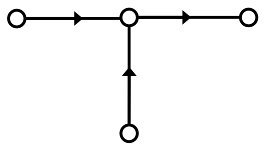
Digital Filter Structures

5.1 The Implementation Problem

Now that we know how to design digital filters and represent them in a variety of ways (difference equation, system function, impulse response), let's turn our attention to implementation. Given a desired filter, how do we implement in hardware or software? The key is to layout the system in the form of a *block* diagram or *signal-flow* diagram.

5.1.1 Components

The components we have to work with are as follows:

Component	Block Diagram	Signal-flow Diagram
Multiplier		
Delay		
Adder		

In this chapter, we will use signal-flow diagrams because they are more compact than block diagrams.

5.1.2 Goals

We will see that there is more than one way to layout a system. The goals of a good layout are:

Complexity: We would like to use the fewest number of components to achieve the desired design.

Stability: A system that is theoretically stable might become unstable when implemented. This is because the filter coefficients must be quantized, so their values are slightly different than anticipated. Each time a coefficient is quantized, there is a small amount of round-off error. For large filters, the round-off error will accumulate to the point that poles might move outside the unit circle.

5.2 Direct-Form I

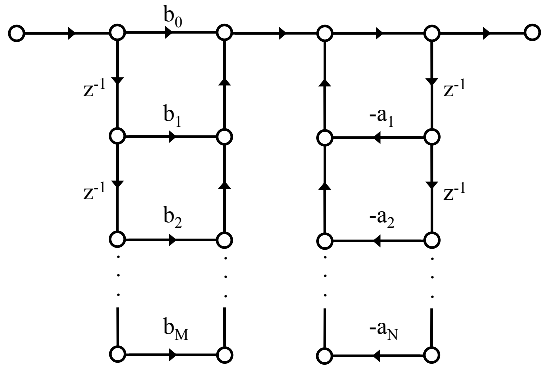
Consider the difference equation:

$$\sum_{k=0}^{N} a_k y[n-k] \quad = \quad \sum_{k=0}^{M} b_k x[n-k]. \tag{5.148}$$

Without loss of generality, set $a_0 = 1$ and solve for $y[n]$,

$$y[n] \quad = \quad \sum_{k=0}^{M} b_k x[n-k] + \sum_{k=1}^{N} (-a_k) y[n-k]. \tag{5.149}$$

This equation can be directly realized with the following signal-flow diagram:



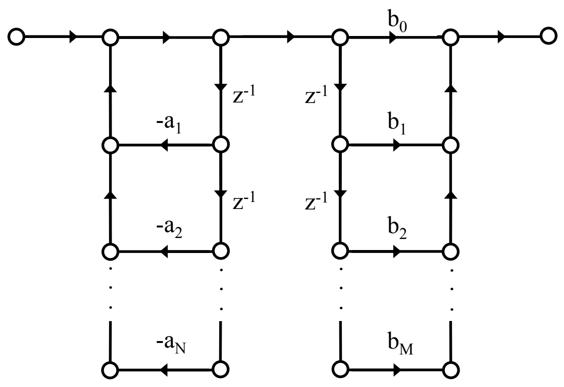
Such a system is called *Direct-Form* I or *DF-I.*

5.3 Direct-Form II

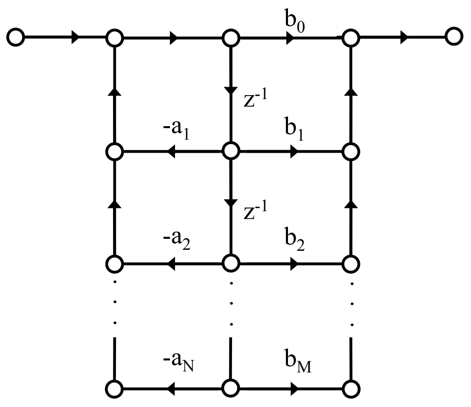
Equation (5.150) can be rearranged as:

$$y[n] \quad = \quad \sum_{k=1}^{N}(-a_k)y[n-k] + \sum_{k=0}^{M} b_k x[n-k]. \tag{5.150}$$

which may be realized as:



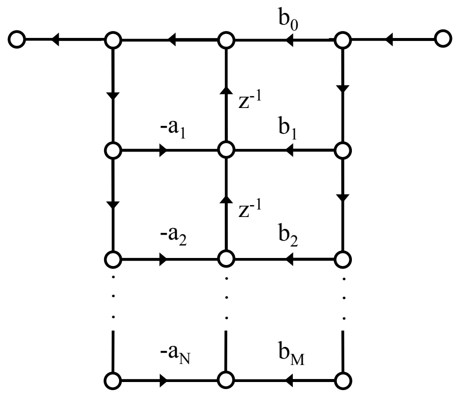
Notice in the figure that the same signal is delayed on both sides of the diagram. Thus, the delays may be combined as follows:



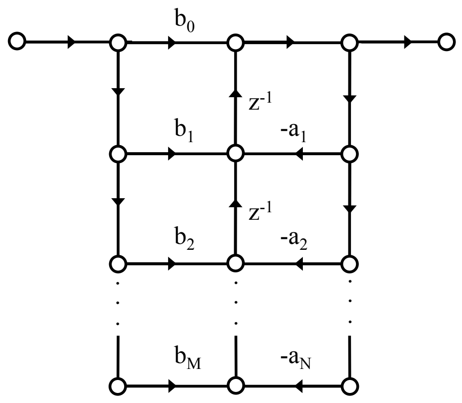
Such a system is called *Direct-Form* II or *DF-II.* The benefit of DF-II over DF-I is that fewer delays are required. For this reason, DF-I is hardly ever used.

5.4 Transposed Form

A signal-flow diagram can be *transposed* by reversing the direction of all arrows in the diagram. For instance, the transposed DF-II structure is as follows:



Notice that when transposing, adders become *branch* nodes and vice versa. In addition, the input and output become reversed. Since the usual convention is to put the input on the left side of the diagram and the output on the right, we can reflect the diagram to get the following representation:



Note that other structures can be transposed, for instance DF-I.

Exercise 5.1: A causal LTI system is represented by the system function:

$$H(z) = \frac{1 - 3z^{-1} + 11z^{-2} - 27z^{-3} + 18z^{-4}}{1 + \frac{3}{4}z^{-1} + \frac{1}{8}z^{-2} - \frac{1}{4}z^{-3} - \frac{1}{16}z^{-4}}.$$

Draw the following structures:

1. Direct-form I.
2. Direct-form II.
3. Transposed direct-form II.

5.5 Cascade Form

Recall that a system function may be put into *factored* form:

$$H(z) \quad = \quad \left(\frac{b_0}{a_0}\right) \frac{\prod_{k=1}^{M}\left(1-\xi_k z^{-1}\right)}{\prod_{k=1}^{N}\left(1-\lambda_k z^{-1}\right)}. \tag{5.151}$$

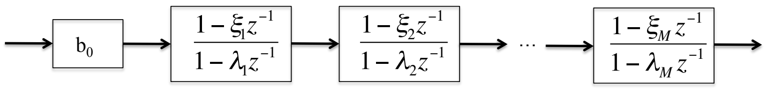
The *cascade* form is an implementation that uses the factored form of the system function.

5.5.1 Cascade of First-Order Sections

For now assume that all poles and zeros are real-valued, that $a_0 = 1$ and that $N = M$. Under these assumptions, equation (5.152) may be expressed as:

$$H(z) \quad = \quad b_0 \prod_{k=1}^{M} \underbrace{\frac{1-\xi_k z^{-1}}{1-\lambda_k z^{-1}}}_{H_k(z)} \tag{5.152}$$

which may be interpreted as the *cascade* of M first order sections. A high-level block diagram is as follows:



b_0
$\frac{1-\xi_1 z^{-1}}{1-\lambda_1 z^{-1}}$
$\frac{1-\xi_2 z^{-1}}{1-\lambda_2 z^{-1}}$
$\cdots$
$\frac{1-\xi_M z^{-1}}{1-\lambda_M z^{-1}}$

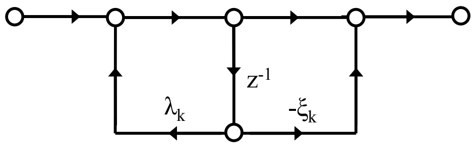
The k^{th} first-order section has transfer function:

$$H_k(z) \quad = \quad \frac{1-\xi_k z^{-1}}{1-\lambda_k z^{-1}} \tag{5.153}$$

and difference equation:

$$y[n] - \lambda_k y[n-1] \quad = \quad x[n] - \xi_k x[n-1]. \tag{5.154}$$

When represented in DF-II form, the k^{th} first-order section (FOS) is as follows:



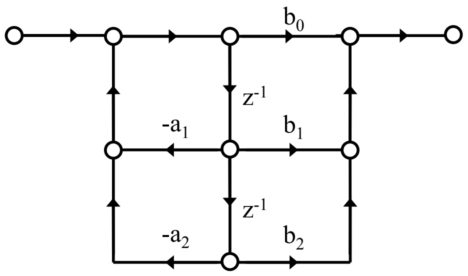
The overall signal-flow diagram is created by cascading M first-order sections, one after the other.

Exercise 5.2: Sketch the cascade form for the system described by the difference equation:

$$\begin{aligned} & y[n] - 1.5y[n-1] + 0.6875y[n-2] - 0.09375y[n-3] \\ = \quad & 2x[n] - 4x[n-1] + 2.8x[n-2] - 0.8x[n-3] + 0.0768x[n-4]. \end{aligned}$$

5.5.2 Cascade of Second-Order Sections

If any of the poles or zeros are complex, then the representation discussed in the previous subsection will require complex multiplications, which is not desirable for a real system. Complex multiplications can be avoided by forming *second-order* sections (SOS's). A DF-II SOS looks like:



The idea is to combine each complex root with its conjugate, which is possible since complex roots always occur in conjugate pairs. To see how to do this, consider a system with a complex conjugate pair of zeros $\{\xi, \xi^*\}$ and a complex conjugate pair of poles $\{\lambda, \lambda^*\}$. In factored form, the system function is:

$$H(z) \quad = \quad \frac{(1-\xi z^{-1})(1-\xi^* z^{-1})}{(1-\lambda z^{-1})(1-\lambda^* z^{-1})}. \tag{5.155}$$

Let's consider just the numerator of (5.156). It can be expanded as

$$(1-\xi z^{-1})(1-\xi^* z^{-1}) \quad = \quad 1-(\xi+\xi^*)z^{-1}+(\xi\xi^*)z^{-2}. \tag{5.156}$$

Since $\xi = \Re\{\xi\} + \Im\{\xi\}$ and $\xi^* = \Re\{\xi\} - \Im\{\xi\}$, if follows that $\xi + \xi^* = 2\Re\{\xi\}$. Furthermore, since $\xi = |\xi|e^{j\angle\xi}$ and $\xi^* = |\xi|e^{-j\angle\xi}$, it follows that $(\xi)(\xi^*) = |\xi|^2$. Thus (5.157) is

$$(1-\xi z^{-1})(1-\xi^* z^{-1}) \quad = \quad 1-2\Re\{\xi\}z^{-1}+|\xi|^2 z^{-2}. \tag{5.157}$$

The numerator of (5.156) can be similarly expanded. Thus, the system function is

$$H(z) \quad = \quad \frac{1-2\Re\{\xi\}z^{-1}+|\xi|^2 z^{-2}}{1-2\Re\{\lambda\}z^{-1}+|\lambda|^2 z^{-2}} \tag{5.158}$$

which is a second-order system with all real-valued coefficients. From (5.159), we see that the coefficients of the SOS are:

$$\begin{aligned} b_0 &= 1 \\ b_1 &= -2\Re\{\xi\} \\ b_2 &= |\xi|^2 \\ -a_1 &= 2\Re\{\lambda\} \\ -a_2 &= -|\lambda|^2. \end{aligned} \tag{5.159}$$

While SOS's are essential for dealing with complex-valued roots, they can also be used when the roots are real. The idea is to combine pairs of real-valued roots to form each SOS. For instance, consider a system with a pair of real-valued zeros $\{\xi_1, \xi_2\}$ and a pair of real-valued poles $\{\lambda_1, \lambda_2\}$. The system function is:

$$\begin{aligned} H(z) &= \frac{(1-\xi_1 z^{-1})(1-\xi_2 z^{-1})}{(1-\lambda_1 z^{-1})(1-\lambda_2 z^{-1})} \\ &= \frac{1-(\xi_1+\xi_2)z^{-1}+\xi_1\xi_2 z^{-2}}{1-(\lambda_1+\lambda_2)z^{-1}+\lambda_1\lambda_2 z^{-2}}. \end{aligned} \tag{5.160}$$

The corresponding set of SOS coefficients are:

$$\begin{aligned} b_0 &= 1 \\ b_1 &= -(\xi_1+\xi_2) \\ b_2 &= \xi_1\xi_2 \\ -a_1 &= (\lambda_1+\lambda_2) \\ -a_2 &= -\lambda_1\lambda_2. \end{aligned} \tag{5.161}$$

It is also possible to have a SOS made up of a pair of real roots and a pair of complex poles, or vice-versa.

Exercise 5.3: A particular causal system has system function:

$$H(z) = \frac{3-9z^{-1}+33z^{-2}-81z^{-3}+54z^{-4}}{1+\frac{3}{4}z^{-1}+\frac{1}{8}z^{-2}-\frac{1}{4}z^{-3}-\frac{1}{16}z^{-4}}.$$

Using a signal-flow graph, draw a cascade structure using second-order direct-form II sections.

5.6 Parallel Form

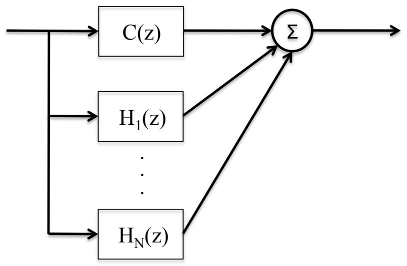
By performing a partial-fraction expansion, a rational system function may be expressed as:

$$H(z) = \underbrace{\sum_{k=0}^{M-N} C_k z^{-k}}_{C(z)} + \sum_{k=1}^{N} \underbrace{\frac{R_k}{1-\lambda_k z^{-1}}}_{H_k(z)}. \tag{5.162}$$

The *parallel* form is an implementation that uses the partial-fraction representation of the system function.

5.6.1 First-Order Sections in Parallel

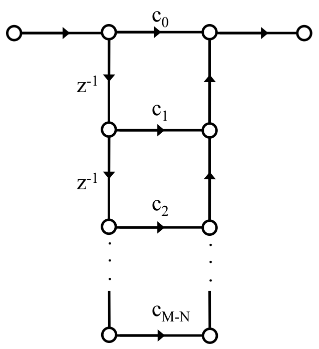
First, let's assume that all poles and residues are are real-valued. Each $H_k(z)$ may be represented as a first-order section with real-valued coefficients. A high-level block diagram of the system is as follows:



The $C(z)$ block may be expanded into a signal flow graph by recalling that:

$$C(z) = \sum_{k=0}^{M-N} C_k z^{-k}. \tag{5.163}$$

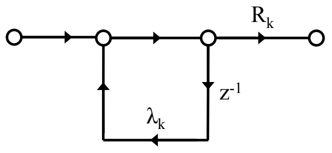
This is merely a FIR system of order $(M-N)$, which may be drawn as:



Each $H_k(z)$ takes on the form

$$H_k(z) = \frac{R_k}{1-\lambda_k z^{-1}} \tag{5.164}$$

which may be drawn as:



Exercise 5.4: A particular causal system has system function:

$$H(z) \quad = \quad \frac{5 - 4z^{-1} + \frac{1}{2}z^{-2}}{1 - \frac{3}{4}z^{-1} + \frac{1}{8}z^{-2}}.$$

Using a signal-flow graph, draw a parallel structure using first-order direct-form II sections.

5.6.2 Second-Order Sections in Parallel

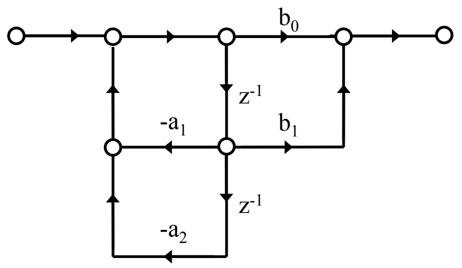
If a pole or its residue is complex, then first-order sections will require complex multiplications, which is not desirable. This can be avoided by joining complex poles and residues with their conjugates to form second-order sections. Let λ be a complex pole and R its corresponding conjugate. We can form a second-order section as follows:

$$\begin{aligned}\frac{R}{1-\lambda z^{-1}}+\frac{R^*}{1-\lambda^* z^{-1}} &= \frac{R\left(1-\lambda^* z^{-1}\right)+R^*\left(1-\lambda z^{-1}\right)}{\left(1-\lambda z^{-1}\right)\left(1-\lambda^* z^{-1}\right)}\\ &= \frac{(R+R^*)-(R\lambda^*+R^*\lambda)z^{-1}}{1-(\lambda+\lambda^*)z^{-1}+\lambda\lambda^* z^{-2}}\\ &= \frac{2\Re\{R\}-(R\lambda^*+R^*\lambda)z^{-1}}{1-2\Re\{\lambda\}z^{-1}+|\lambda|^2 z^{-2}} \qquad (5.165)\end{aligned}$$

where

$$R\lambda^*+R^*\lambda = 2|R||\lambda|\cos\left(\angle R-\angle\lambda\right). \qquad (5.166)$$

This is a second-order section of the form:



where

$$\begin{aligned} b_0 &= 2\Re\{R\}\\ b_1 &= -2|R||\lambda|\cos\left(\angle R-\angle\lambda\right)\\ -a_1 &= 2\Re\{\lambda\}\\ -a_2 &= -|\lambda|^2. \qquad (5.167)\end{aligned}$$

Note that second-order sections can be used even when poles and residues are real-valued. Consider a pair of real-valued poles $\{\lambda_1,\lambda_2\}$ and their corresponding residues $\{R_1,R_2\}$. We can combine them to form the second-order section:

$$\begin{aligned}\frac{R_1}{1-\lambda_1 z^{-1}}+\frac{R_2}{1-\lambda_2 z^{-1}} &= \frac{R_1(1-\lambda_2 z^{-1})+R_2(1-\lambda_1 z^{-1})}{(1-\lambda_1 z^{-1})(1-\lambda_2 z^{-1})}\\ &= \frac{(R_1+R_2)-(R_1\lambda_2+R_2\lambda_1)z^{-1}}{1-(\lambda_1+\lambda_2)z^{-1}+\lambda_1\lambda_2 z^{-2}}. \qquad (5.168)\end{aligned}$$

The coefficients of the corresponding SOS are:

$$\begin{aligned} b_0 &= R_1+R_2\\ b_1 &= -(R_1\lambda_2+R_2\lambda_1)\\ -a_1 &= (\lambda_1+\lambda_2)\\ -a_2 &= -\lambda_1\lambda_2. \qquad (5.169)\end{aligned}$$

Exercise 5.5: A particular causal system has system function:

$$H(z) = \frac{3 - 9z^{-1} + 33z^{-2} - 81z^{-3} + 54z^{-4}}{1 + \frac{3}{4}z^{-1} + \frac{1}{8}z^{-2} - \frac{1}{4}z^{-3} - \frac{1}{16}z^{-4}}.$$

Using a signal-flow graph, draw a parallel structure using second-order direct-form II sections.

5.7 Problems

1. A particular discrete-time system can be represented by the following difference equation:

$$y[n] + \frac{1}{2}y[n-1] - \frac{3}{16}y[n-2] \quad = \quad x[n] + x[n-1] + \frac{1}{4}x[n-2]$$

   For this system:

   (a) Determine the impulse response.

   (b) Using a signal flow graph, draw the direct form I structure.

   (c) Using a signal flow graph, draw the direct form II structure.

   (d) Using a signal flow graph, draw the transposed direct form II structure.

   (e) Using a signal flow graph, draw a cascade structure using first-order direct form II sections.

   (f) Using a signal flow graph, draw a parallel structure using first-order direct form II sections.

2. A particular discrete-time system can be represented by the following difference equation:

$$y[n] - 0.5y[n-1] + 0.1y[n-2] + 0.3y[n-3] - 0.0936y[n-4] \quad =$$

$$5x[n] + 16.5x[n-1] + 14.7x[n-2] - 22.04x[n-3] - 33.6x[n-4]$$

   For this system:

   (a) Determine the real-valued impulse response. Although this system has complex-valued poles, your answer should not contain any imaginary values.

   (b) Using a signal flow graph, draw the direct form I structure.

   (c) Using a signal flow graph, draw the direct form II structure.

   (d) Using a signal flow graph, draw the transposed direct form II structure.

   (e) Using a signal flow graph, draw a cascade structure using second-order direct form II sections. Make sure your system does not require any complex multiplications.

   (f) Using a signal flow graph, draw a parallel structure using second-order direct form II sections. Make sure your system does not require any complex multiplications.

Chapter 6

The FFT

6.1 Motivation for Finite Transforms

Recall the expression for the DTFT:

$$X\left(e^{j\omega}\right) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}. \tag{6.170}$$

To compute the DTFT numerically (for instance, in Matlab), we had to sample the frequency axis over a finite range of ω. In order to minimize complexity and storage requirements, we would like the number of frequency samples to be as small as possible. Let's now consider the problem of picking the number of frequency samples in an intelligent way.

6.2 Discrete Fourier Series

Let $\tilde{x}[n]$ be a *periodic* sequence with fundamental period N,

$$\tilde{x}[n] = \tilde{x}[n+kN] \tag{6.171}$$

for all k and all n. Let $x[n], 0 \leq n \leq N-1$ be one period (the *fundamental* period) of the sequence $\tilde{x}[n]$. Then $\tilde{x}[n]$ can be represented by the discrete Fourier series (DFS)

$$\tilde{x}[n] = \frac{1}{N}\sum_{k=0}^{N-1} \tilde{X}[k]W_N^{-nk} \tag{6.172}$$

where

$$W_N = e^{-j2\pi/N} \tag{6.173}$$

is called the n^{th} *root of unity*, and

$$\tilde{X}[k] = \sum_{n=0}^{N-1} \tilde{x}[n]W_N^{nk} \tag{6.174}$$

are the DFS coefficients. Just as $\tilde{x}[n]$ is periodic (with period N), so is $\tilde{X}[k]$ (also with period N).

Exercise 6.1: Find the DFS coefficients for the sequence $\tilde{x}[n]$ with fundamental period:

$$x[n] = \left\{ \underset{\uparrow}{3}, 2, 1, 0 \right\}.$$

6.3 Discrete Fourier Transform

Suppose that instead of a periodic sequence $\tilde{x}[n]$, you have a finite sequence $x[n]$ which is zero outside the range $0 \leq n \leq N-1$. We can still find a DFS representation by "forcing" the signal to be periodic. This is done by defining the *periodic extension* of $x[n]$ to be

$$\tilde{x}[n] = x[n \mod N]. \tag{6.175}$$

The *discrete Fourier transform* (DFT) of the finite signal $x[n]$ is found from the DFS of its periodic extension (in particular, the DFT is the fundamental period of the DFS of $\tilde{x}[n]$). The DFT coefficients are:

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}, \tag{6.176}$$

which is a finite sequence defined over $0 \leq k \leq N-1$. The inverse DFT is:

$$x[n] = \frac{1}{N} \sum_{k=0}^{N-1} X[k] W_N^{-nk}. \tag{6.177}$$

As an example, the DFT of the sequence at the top of this page are the coefficients $X[k]$ that form the fundamental period of $\tilde{X}[k]$.

6.3.1 Relationship with DTFT

If you compare the equation for the DFT (6.177) with that of the DTFT (6.171), you will see that

$$\begin{aligned} X[k] &= X\left(e^{j\omega}\right)\Big|_{\omega=k(2\pi/N)} \\ &= \sum_{n=-\infty}^{\infty} x[n]e^{-jnk(2\pi/N)} \\ &= \sum_{n=0}^{N-1} x[n]\left(e^{-j(2\pi/N)}\right)^{nk} \\ &= \sum_{n=0}^{N-1} x[n]W_N^{nk} \end{aligned} \qquad (6.178)$$

where the summation is only over $0 \leq n \leq N-1$ since $x[n]$ is zero outside that range of n. Thus, we may conclude that the DFT is merely the DTFT sampled at a frequency spacing of $2\pi/N$.

6.3.2 Matlab Implementation

We can leverage our previous *dtft* function to come up with a *dft* function. Rather than passing *omega* in as an argument, we will have the program itself determine an appropriate frequency axis. Furthermore, rather than passing in n as an argument, we will assume that n ranges from 0 to $N-1$, where N is the length of the sequence $x[n]$.

```
function X = dft( x )
% Computes the DFT
% X = dft( x )
% where x = sampled signal defined over 0 <= n <= N-1
%       X = the DFT coefficients
N = length( x );
n = 0:(N-1);
omega = n*2*pi/N;
X = dtft( x, n, omega );
```

6.4 The FFT Algorithm

The *fast Fourier transform* is an algorithm for efficiently computing the DFT. It performs best when N is a power of 2. Matlab has a built-in *fft* function. As an exercise, compare the speed of our *dft* function against that of Matlab's *fft* function for a $N = 2^{12} = 4,096$ point transform.

```
N = 2^10;
tic;dft(rand(1,N));toc
tic;fft(rand(1,N));toc
```

Now compare the speed of *fft* for a $N = 2^{20} = 1,048,576$ point transform against that of a $N = 2^{20} - 3 = 1,048,573$ point transform:

```
N = 2^20;
tic;fft(rand(1,N));toc
tic;fft(rand(1,N-3));toc
```

As you can see, *fft* is not as efficient when N is not a power of 2, but it is still more efficient than our *dft* function.

6.5 Circular Convolution

Consider two finite sequences $x[n], y[n]$ and their periodic extensions $\tilde{x}[n], \tilde{y}[n]$. From the convolution theorem of the Fourier transform, it follows that:

$$\mathcal{F}\left\{\tilde{x}[n] * \tilde{y}[n]\right\} = \mathcal{F}\left\{\tilde{x}[n]\right\}\mathcal{F}\left\{\tilde{y}[n]\right\}. \tag{6.179}$$

Because the DFT is the DTFT of the periodic extension, it follows that

$$\mathcal{F}\left\{\tilde{x}[n] * \tilde{y}[n]\right\} = X[k]Y[k]. \tag{6.180}$$

At first glance, this appears to have the usual relationship: That convolution in the time domain corresponds to multiplication in the frequency domain. However, notice that we are not convolving the two finite-length sequences $x[n], y[n]$, but instead we are convolving their periodic extensions $\tilde{x}[n], \tilde{y}[n]$. Such an operation is called a *circular convolution*. In matlab, circular convolution is implemented with the *cconv* function.

Exercise 6.2: Circularly convolve the following sequences:

$$\begin{aligned} x[n] &= \left\{ \underset{\uparrow}{1}, 2, 0, 1 \right\} \\ y[n] &= \left\{ \underset{\uparrow}{2}, 2, 1, 1 \right\} \end{aligned}$$

Do this in both the time domain (using *cconv*) and in the frequency domain (using *fft* and *ifft*).

6.5.1 Linear Convolution Using the FFT

Recall normal linear convolution of two finite sequences, i.e. $x[n] * y[n]$. If $x[n]$ has a length of N_1 and $y[n]$ has a length of N_2, then the result of the convolution will have a length of $N = N_1 + N_2 - 1$. It follows that linear convolution can be implemented from circular convolution (and hence, the fft), if both sequences are zero padded out to a length of N.

Exercise 6.3: Using the FFT and zero-padding, linearly convolve the sequences given in the last exercise.

6.5.2 Filtering with the FFT

The FFT gives an alternative way to implement a digital filter. The idea is that you first take the FFT of the input sequence. Next, you multiply the transformed signal by the desired frequency response. Finally, you take the inverse FFT.

Exercise 6.4: In this example, we have a signal $x[n] = ne^{-0.03n}, 0 \leq n \leq 255$, corrupted by noise, which we would like to filter:

```
n = 0:255;
x = n.*exp(-0.03*n);  % desired signal
y = x + randn(size(x));   % noisy signal
figure(1);plot(n,x);  % let’s use plot instead of stem
figure(2);plot(n,y);  % since it is a long sequence
X = fft(x);
Y = fft(y);
figure(3);plot( abs(X) );
figure(4);plot( abs(Y) );
H = [ones(1,15) zeros(1,226), ones(1,15)];  % the filter response
Z = Y.*H; % filter in frequency domain
z = ifft( Z );
figure(5);plot(n, real(z) );
```

6.6 Problems

1. Determine the DFS coefficients of the following periodic sequences using the DFS definition,

   (a) $\tilde{x}_1[n] = \{2, 0, 2, 0\}, N = 4.$

   (b) $\tilde{x}_2[n] = \{0, 0, 1, 0, 0\}, N = 5.$

   (c) $\tilde{x}_3[n] = \{3, -3, 3, -3\}, N = 4.$

   (d) $\tilde{x}_4[n] = \{j, j, -j, -j\}, N = 4.$

   This problem should be done with paper-and-pencil, but you may verify your answers in matlab.

2. *Note: this problem should be done in matlab.* A 12 point sequence $x[n]$ is defined as

$$x[n] \quad = \quad \{1, 2, 3, 4, 5, 6, 6, 5, 4, 3, 2, 1\}$$

   (a) Determine the DFT $X(k)$ of $x[n]$. Plot (using the *stem* function) its magnitude and phase.

   (b) Plot the magnitude and phase of the DTFT $X(e^{j\omega})$ of $x[n]$ using matlab.

   (c) Verify that the above DFT is the sampled version of $X(e^{j\omega})$. It might be helpful to combine the above two plots in one graph using the *hold* command.