CS2403- DIGITAL SIGNAL PROCESSING

UNIT I

Signals:

Signal is a function of one or more independent variables which contain some information.

System:

A system is a set of elements or functional block that are connected together and produces an output in response to an input signal.

Processing:

Operations performed by the system on the signal is called processing.

TypesOf signal processing:

1.Analog signal processing.
2.Digital signal processing.

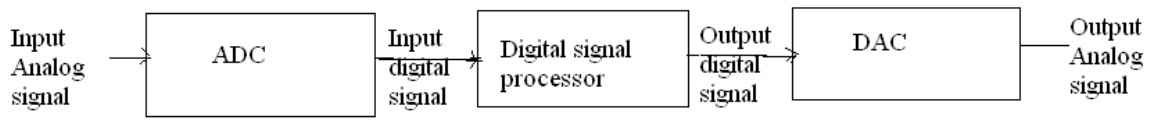
Analog signal processing:

In this processing input signal,output signal and the system are all analog in nature.

Digital signal processing:

Dsp is the processing of signals by digital systems.Input and output signals are also digital in nature.

Basic elements of Digital signal processing:



ADC:

It converts analog input to digital input.

Digital Signal processor:

It performs amplification,attenuation,filtering,spectral analysis,feature extraction etc operations on digital data.

DAC:

Some of the processed signals are required back in the analog form.So we use DAC to convert digital output to its analog equivalent.

Advantages of Dsp over Anlog signal processing:

1.Flexibility.
2.Accuracy.
3.Easy storage.
4.Mathematical processing.
5.Cost.

Applications:

1.Speech processing.
2.Telecommunication.
3.Biomedical Engineering.
4.Instrumentation

Types of signals based on time:

1.Continuous time signals.
2.Discrete time signals.

Continuous time signals:

Continuous time or analog signals are signals that are defined for every value of $a < t < b$, where (a, b) can be (-¥, + ¥), i.e., $x(t) = e^{-|t|}$ or $x(t) = \cos(pt)$. .

Frequency concept in continuous time signal:

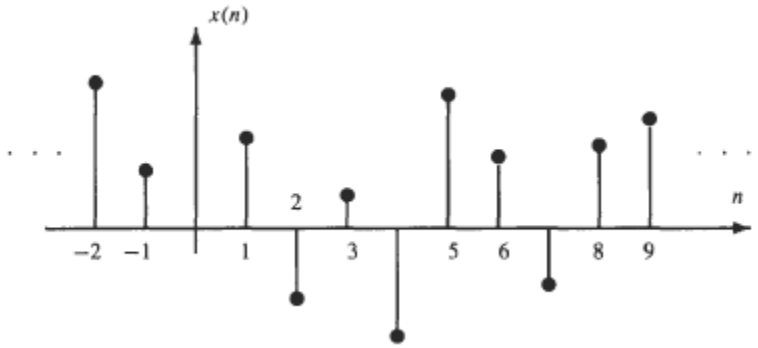
$$x_a(t) = A\cos(\Omega t + \theta) \qquad -\infty < t < \infty$$

$x_a(t + T_p) = x_a(t)$, $T_p = \frac{1}{F}$: fundamental period. Increasing F means increasing oscillation in time domain. $F = 0$ corresponds to $Tp = \infty$.

Discrete time signal:

Discrete-time signals are defined at discrete-time instants and between the two discrete time instants are undefined but are not zero. They can be obtained either by sampling analog signals or they can be discrete in nature like discrete measurement signals.

A discrete-time signal having a set of discrete values is called a digital signal. Note that sampling an analog signal produces a discrete-time signal. Then quantization of its values produces a digital signal.



Thus, the sequence values x(0) to x(N - 1) may often be considered to be the elements of a column vector as follows:

$$\mathbf{x} = [x(0), x(1), \ldots, x(N-1)]^T$$

Discrete-time signals are often derived by sampling a continuous-time signal, such as speech, with an analogto-digital (A/D) converter) For example, a continuous-time signal x,(t) that is sampled at a rate of fs = l/Ts samples per second produces the sampled signal x(n), which is related to xa(t) as follows:

$$x(n) = x_0(nT_s)$$

Classification of Signals

1. Finite duration $x(n) = 0 \quad n > N$

   Infinite duration

2. Right-Left sided

   $x(n) = 0 \qquad n < N_1$ right-sided

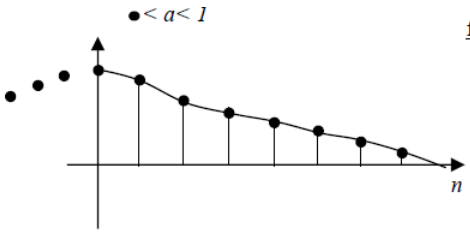
   $x(n) = 0 \qquad n > N_2$ left-sided

Some Elementary Discrete-Time Signals

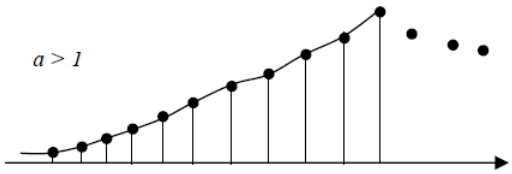
- unit sample sequence $\sigma(n) = \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases}$
- unit step sequence $u(n) = \begin{cases} 1 & n \geq 0 \\ 0 & n < 0 \end{cases}$
- unit ramp $u_r(n) = \begin{cases} n & n \geq 0 \\ 0 & n < 0 \end{cases}$
- exponential $x(n) = a^n$ for all n

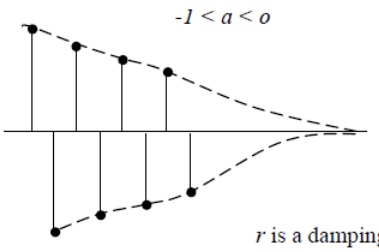
If a is complex, then $a = re^{j\theta} \rightarrow x(n) = r^n e^{j\theta n}$

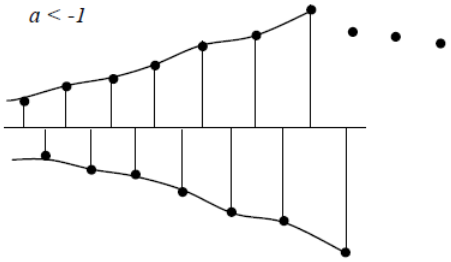
$$= r^n (\cos\theta n + j \sin\theta n)$$



for real *a*







r is a damping factor

Energy and Power Signals

Energy is defined as $E = \sum_{-\infty}^{+\infty} |x(n)|^2$ if E is finite, i.e., $o < E < \infty$, then $x(n)$ is called Energy Signal. However, many signals that have an infinite energy, have a finite average power. Average power is defined as

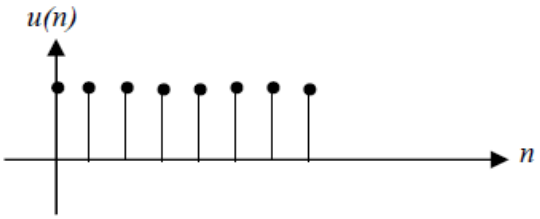
$$P_{ave} = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{-N}^{N} |x(n)|^2 .$$

If we define the signal energy of $x(n)$ over the interval $(-N, N)$ as

$$E_N = \sum_{-N}^{N} |x(n)|^2 \quad \text{then} \quad E = \lim_{N\to\infty} E_n$$

and therefore, $P_{ave} = \lim_{N\to\infty} \frac{1}{2N+1} E_N$ clearly if E is finite, then $P_{ave} = 0$.

Example – Unit Step Sequence



Obviously, it is not an energy signal but it is a power signal.

$$p = \lim_{N\to\infty} \frac{1}{2N+1} \sum_{-N}^{N} |x(n)|^2 = \lim \frac{1}{2N+1} \sum_{n=0}^{N} (1)^2$$

$$= \lim_{N\to\infty} \frac{N+1}{2N+1} = \frac{1}{2} \Rightarrow \quad \text{it is a power signal!}$$

Periodic and Aperiodic Sequences:

A discrete-time signal may always be classified as either being periodic or aperiodic. A signal x(n) is said to be periodic if, for some positive real integer *N*,

$$X(n)=x(n+N) \qquad (1.1)$$

for all n. This is equivalent to saying that the sequence repeats itself every N samples. If a signal is periodic with
period N, it is also periodic with period 2N, period 3N, and all other integer multiples of N. The fundamental period, which we will denote by N, is the smallest positive integer.
If Eq. (1 .I) is not satisfied for any integer N, x(n) is said to be an aperiodic signal.

EXAMPLE 1.2.1 The signals

$$x_1(n) = a^n u(n) = \begin{cases} a^n & n \geq 0 \\ 0 & n < 0 \end{cases}$$

and

$$x_2(n) = \cos(n^2)$$

are not periodic, whereas the signal

$$x_3(n) = e^{j\pi n/8}$$

is periodic and has a fundamental period of $N = 16$.

If $x_1(n)$ is a sequence that is periodic with a period N_1, and $x_2(n)$ is another sequence that is periodic with a period N_2, the sum

$$x(n) = x_1(n) + x_2(n)$$

will always be periodic and the fundamental period is

$$N = \frac{N_1 N_2}{\gcd(N_1, N_2)} \qquad (1.2)$$

where $\gcd(N_1, N_2)$ means *the greatest common divisor* of N_1 and N_2. The same is true for the product; that is,

$$x(n) = x_1(n)x_2(n)$$

will be periodic with a period N given by Eq. (*1.2*). However, the fundamental period may be smaller.

Given any sequence $x(n)$, a periodic signal may always be formed by replicating $x(n)$ as follows:

$$y(n) = \sum_{k=-\infty}^{\infty} x(n - kN)$$

where N is a positive integer. In this case, $y(n)$ will be periodic with period N.

1.2.5 *Symmetric Sequences*

A discrete-time signal will often possess some form of symmetry that may be exploited in solving problems. Two symmetries of interest are as follows:

Definition: A real-valued signal is said to be *even* if, for all n,

$$x(n) = x(-n)$$

whereas a signal is said to be *odd* if, for all n,

$$x(n) = -x(-n)$$

Any signal $x(n)$ may be decomposed into a sum of its even part, $x_e(n)$, and its odd part, $x_o(n)$, as follows:

$$x(n) = x_e(n) + x_o(n) \tag{1.3}$$

To find the even part of $x(n)$ we form the sum

$$x_e(n) = \tfrac{1}{2}\{x(n) + x(-n)\}$$

whereas to find the odd part we take the difference

$$x_o(n) = \tfrac{1}{2}\{x(n) - x(-n)\}$$

For complex sequences the symmetries of interest are slightly different.

Definition: A complex signal is said to be *conjugate symmetric*[3] if, for all n,

$$x(n) = x^*(-n)$$

and a signal is said to be *conjugate antisymmetric* if, for all n,

$$x(n) = -x^*(-n)$$

Any complex signal may always be decomposed into a sum of a conjugate symmetric signal and a conjugate antisymmetric signal.

1.2.6 *Signal Manipulations*

In our study of discrete-time signals and systems we will be concerned with the manipulation of signals. These manipulations are generally compositions of a few basic signal transformations. These transformations may be classified either as those that are transformations of the independent variable n or those that are transformations of the amplitude of $x(n)$ (i.e., the dependent variable). In the following two subsections we will look briefly at these two classes of transformations and list those that are most commonly found in applications.

Transformations of the Independent Variable

Sequences are often altered and manipulated by modifying the index n as follows:

$$y(n) = x(f(n))$$

where $f(n)$ is some function of n. If, for some value of n, $f(n)$ is not an integer, $y(n) = x(f(n))$ is undefined. Determining the effect of modifying the index n may always be accomplished using a simple tabular approach of listing, for each value of n, the value of $f(n)$ and then setting $y(n) = x(f(n))$. However, for many index transformations this is not necessary, and the sequence may be determined or plotted directly. The most common transformations include shifting, reversal, and scaling, which are defined below.

Shifting This is the transformation defined by $f(n) = n - n_0$. If $y(n) = x(n - n_0)$, $x(n)$ is shifted to the right by n_0 samples if n_0 is positive (this is referred to as a delay), and it is shifted to the left by n_0 samples if n_0 is negative (referred to as an advance).

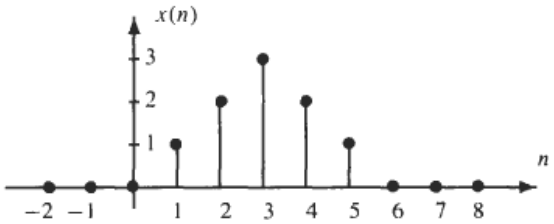
Reversal This transformation is given by $f(n) = -n$ and simply involves "flipping" the signal $x(n)$ with respect to the index n.

Time Scaling This transformation is defined by $f(n) = Mn$ or $f(n) = n/N$ where M and N are positive integers. In the case of $f(n) = Mn$, the sequence $x(Mn)$ is formed by taking every Mth sample of $x(n)$ (this operation is known as *down-sampling*). With $f(n) = n/N$ the sequence $y(n) = x(f(n))$ is defined as follows:

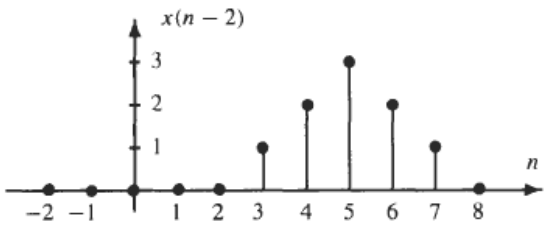
$$y(n) = \begin{cases} x\left(\dfrac{n}{N}\right) & n = 0, \pm N, \pm 2N, \cdots \\ 0 & \text{otherwise} \end{cases}$$

(this operation is known as *up-sampling*).

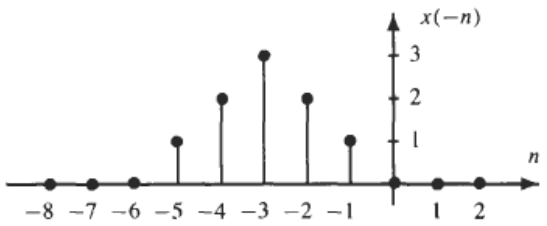
Examples of shifting, reversing, and time scaling a signal are illustrated in Fig. 1-2.



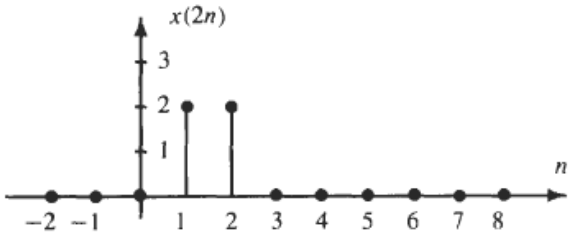
(*a*) A discrete-time signal.



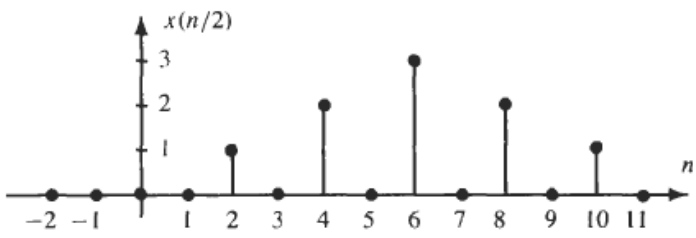
(*b*) A delay by $n_0 = 2$.



(*c*) Time reversal.

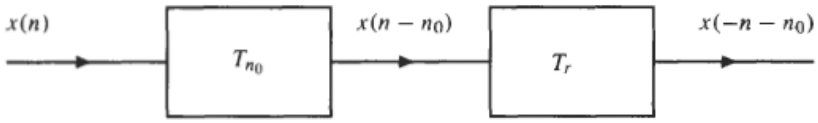


(*d*) Down-sampling by a factor of 2.

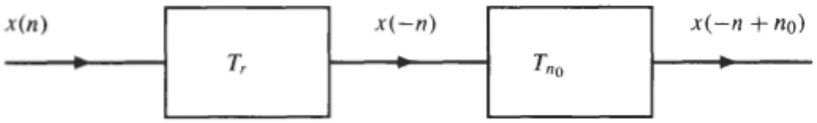


(*e*) Up-sampling by a factor of 2.

Shifting, reversal, and time-scaling operations are order-dependent. Therefore, one needs to be careful in evaluating compositions of these operations. For example, Fig. 1-3 shows two systems, one that consists of a delay followed by a reversal and one that is a reversal followed by a delay. As indicated, the outputs of these two systems are not the same.



(*a*) A delay T_{n_0} followed by a time-reversal T_r.



(*b*) A time-reversal T_r followed by a delay T_{n_0}.

Fig. 1-3. Example illustrating that the operations of delay and reversal do not commute.

Addition, Multiplication, and Scaling

The most common types of amplitude transformations are addition, multiplication, and scaling. Performing these operations is straightforward and involves only pointwise operations on the signal.

Addition The sum of two signals

$$y(n) = x_1(n) + x_2(n) \qquad -\infty < n < \infty$$

is formed by the pointwise addition of the signal values.

Multiplication The multiplication of two signals

$$y(n) = x_1(n)x_2(n) \qquad -\infty < n < \infty$$

is formed by the pointwise product of the signal values.

Scaling Amplitude scaling of a signal $x(n)$ by a constant c is accomplished by multiplying every signal value by c:

$$y(n) = cx(n) \qquad -\infty < n < \infty$$

This operation may also be considered to be the product of two signals, $x(n)$ and $f(n) = c$.

1.3 DISCRETE-TIME SYSTEMS

A discrete-time system is a mathematical operator or mapping that transforms one signal (the input) into another signal (the output) by means of a fixed set of rules or operations. The notation $T[\cdot]$ is used to represent a general system as shown in Fig. 1-4, in which an input signal $x(n)$ is transformed into an output signal $y(n)$ through the transformation $T[\cdot]$. The input-output properties of a system may be specified in any one of a number of different ways. The relationship between the input and output, for example, may be expressed in terms of a concise mathematical rule or function such as

$$y(n) = x^2(n)$$

or

$$y(n) = 0.5y(n-1) + x(n)$$

It is also possible, however, to describe a system in terms of an algorithm that provides a sequence of instructions or operations that is to be applied to the input signal, such as

$$\begin{aligned} y_1(n) &= 0.5y_1(n-1) + 0.25x(n) \\ y_2(n) &= 0.25y_2(n-1) + 0.5x(n) \\ y_3(n) &= 0.4y_3(n-1) + 0.5x(n) \\ y(n) &= y_1(n) + y_2(n) + y_3(n) \end{aligned}$$

In some cases, a system may conveniently be specified in terms of a table that defines the set of all possible input-output signal pairs of interest.

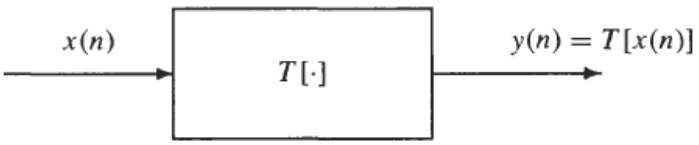


Fig. 1-4. The representation of a discrete-time system as a transformation $T[\cdot]$ that maps an input signal $x(n)$ into an output signal $y(n)$.

Discrete-time systems may be classified in terms of the properties that they possess. The most common properties of interest include linearity, shift-invariance, causality, stability, and invertibility. These properties, along with a few others, are described in the following section.

1.3.1 System Properties

Memoryless System

The first property is concerned with whether or not a system has memory.

> **Definition:** A system is said to be memoryless if the output at any time $n = n_0$ depends only on the input at time $n = n_0$.

In other words, a system is memoryless if, for any n_0, we are able to determine the value of $y(n_0)$ given only the value of $x(n_0)$.

EXAMPLE 1.3.1 The system

$$y(n) = x^2(n)$$

is memoryless because $y(n_0)$ depends only on the value of $x(n)$ at time n_0. The system

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

on the other hand, is not memoryless because the output at time n_0 depends on the value of the input both at time n_0 and at time $n_0 - 1$.

Static versus Dynamic Systems

Static Systems ≡ memory less ≡ the output doesn't depend on past or future values of the input.

Dynamic Systems ≡ having either infinite or finite memory.

Example: $y(n) = 2x(n) + x(n)^2$ Static

$$y(n) = \sum_{k=0}^{N} x(n-k) \text{ Dynamic-finite}$$

$$y(n) = \sum_{k=0}^{\infty} x(n-k) \text{ Dynamic-infinite}$$

Time invariant versus Time-Invariant Systems

A relaxed system Γ is time-invariant if

$$x(n) \rightarrow y(n)$$

$$\forall K, x(n-k) \rightarrow y(n-k)$$

Example: 1) $y(n) = x(n) - x\,(n-1)$

$y(n-k) = x(n-k) - x(n-k-1)$ time invariant

2) $y(n) = nx(n)$

$y(n-k) = (n-k)\,x(n-k) = nx(n-k) - kx(n-k)$

but $x(n-k) \rightarrow nx(n-k) \neq y(n-k)$

$\Rightarrow$ time variant

Causality

A system is causal if the output at any time depends only on present and past values of the inputs and not on future values of the input. $y(n) = x(-n)$ is non-casual because $y(-1) = x(1)$!

Stable versus Unstable Systems

A system is *Stable* if any bounded input produces bounded output (BIBO).

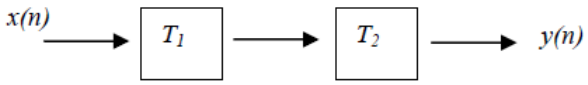
Otherwise, it is *unstable*.

Linearity

A system is linear if

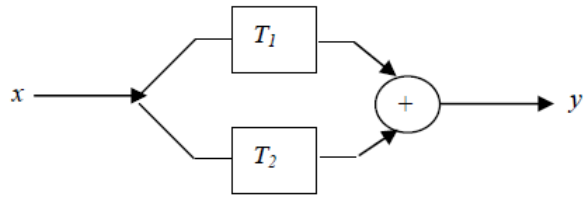
$$T(a_1x_1(n) + a_2x_2(n)) = a_1T[x_1(n)] + a_2T[x_2(n)]$$

two systems can be connected to each other in two ways:



$T = T_1 \bullet T_2 = T_2 \bullet T_1$

= for LTI systems only



$T = T_1 + T_2$

Causality

A system property that is important for real-time applications is *causality*, which is defined as follows:

> **Definition:** A system is said to be *causal* if, for any n_0, the response of the system at time n_0 depends only on the input up to time $n = n_0$.

For a causal system, changes in the output cannot precede changes in the input. Thus, if $x_1(n) = x_2(n)$ for $n \leq n_0$, $y_1(n)$ must be equal to $y_2(n)$ for $n \leq n_0$. Causal systems are therefore referred to as *nonanticipatory*. An LSI system will be causal if and only if $h(n)$ is equal to zero for $n < 0$.

EXAMPLE 1.3.4 The system described by the equation $y(n) = x(n) + x(n-1)$ is causal because the value of the output at any time $n = n_0$ depends only on the input $x(n)$ at time n_0 and at time $n_0 - 1$. The system described by $y(n) = x(n) + x(n+1)$, on the other hand, is noncausal because the output at time $n = n_0$ depends on the value of the input at time $n_0 + 1$.

Stability

In many applications, it is important for a system to have a response, $y(n)$, that is bounded in amplitude whenever the input is bounded. A system with this property is said to be *stable* in the bounded input-bounded output (BIBO) sense. Specifically,

> **Definition:** A system is said to be stable in the bounded input-bounded output sense if, for any input that is bounded, $|x(n)| \leq A < \infty$, the output will be bounded,
>
> $$|y(n)| \leq B < \infty$$

For a linear shift-invariant system, stability is guaranteed if the unit sample response is absolutely summable:

$$\sum_{n=-\infty}^{\infty} |h(n)| < \infty \tag{1.8}$$

EXAMPLE 1.3.5 An LSI system with unit sample response $h(n) = a^n u(n)$ will be stable whenever $|a| < 1$, because

$$\sum_{n=-\infty}^{\infty} |h(n)| = \sum_{n=0}^{\infty} |a|^n = \frac{1}{1-|a|} \qquad |a| < 1$$

The system described by the equation $y(n) = nx(n)$, on the other hand, is not stable because the response to a unit step, $x(n) = u(n)$, is $y(n) = nu(n)$, which is unbounded.

1.4 CONVOLUTION

The relationship between the input to a linear shift-invariant system, $x(n)$, and the output, $y(n)$, is given by the convolution sum

$$x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k)$$

Because convolution is fundamental to the analysis and description of LSI systems, in this section we look at the mechanics of performing convolutions. We begin by listing some properties of convolution that may be used to simplify the evaluation of the convolution sum.

1.4.1 Convolution Properties

Convolution is a linear operator and, therefore, has a number of important properties including the commutative, associative, and distributive properties. The definitions and interpretations of these properties are summarized below.

Commutative Property

The commutative property states that the order in which two sequences are convolved is not important. Mathematically, the commutative property is

$$x(n) * h(n) = h(n) * x(n)$$

From a systems point of view, this property states that a system with a unit sample response $h(n)$ and input $x(n)$ behaves in exactly the same way as a system with unit sample response $x(n)$ and an input $h(n)$. This is illustrated in Fig. 1-5(a).

Associative Property

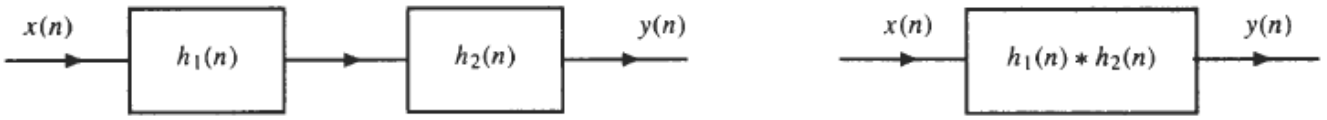
The convolution operator satisfies the associative property, which is

$$\{x(n) * h_1(n)\} * h_2(n) = x(n) * \{h_1(n) * h_2(n)\}$$

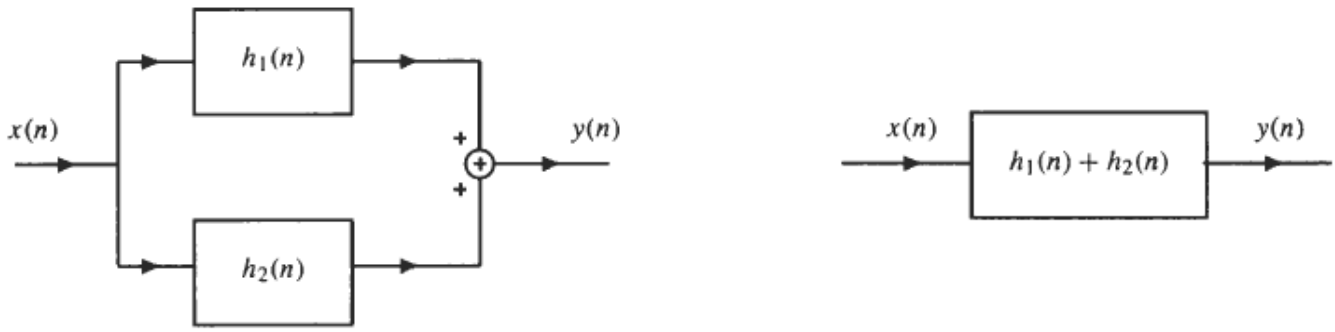
From a systems point of view, the associative property states that if two systems with unit sample responses $h_1(n)$ and $h_2(n)$ are connected in cascade as shown in Fig. 1-5(b), an equivalent system is one that has a unit sample response equal to the convolution of $h_1(n)$ and $h_2(n)$:

$$h_{eq}(n) = h_1(n) * h_2(n)$$

(a) The commutative property.



(b) The associative property.



(c) The distributive property.

Distributive Property

The distributive property of the convolution operator states that

$$x(n) * \{h_1(n) + h_2(n)\} = x(n) * h_1(n) + x(n) * h_2(n)$$

From a systems point of view, this property asserts that if two systems with unit sample responses $h_1(n)$ and $h_2(n)$ are connected in parallel, as illustrated in Fig. 1-5(c), an equivalent system is one that has a unit sample response equal to the sum of $h_1(n)$ and $h_2(n)$:

$$h_{eq}(n) = h_1(n) + h_2(n)$$

1.4.2 Performing Convolutions

Having considered some of the properties of the convolution operator, we now look at the mechanics of performing convolutions. There are several different approaches that may be used, and the one that is the easiest will depend upon the form and type of sequences that are to be convolved.

Direct Evaluation

When the sequences that are being convolved may be described by simple closed-form mathematical expressions, the convolution is often most easily performed by directly evaluating the sum given in Eq. (*1.7*). In performing convolutions directly, it is usually necessary to evaluate finite or infinite sums involving terms of the form α^n or $n\alpha^n$. Listed in Table 1-1 are closed-form expressions for some of the more commonly encountered series.

EXAMPLE 1.4.1 Let us perform the convolution of the two signals

$$x(n) = a^n u(n) = \begin{cases} a^n & n \geq 0 \\ 0 & n < 0 \end{cases}$$

and

$$h(n) = u(n)$$

With the direct evaluation of the convolution sum we find

$$y(n) = x(n) * h(n) = \sum_{k=-\infty}^{\infty} x(k)h(n-k) = \sum_{k=-\infty}^{\infty} a^k u(k)u(n-k)$$

Because $u(k)$ is equal to zero for $k < 0$ and $u(n-k)$ is equal to zero for $k > n$, when $n < 0$, there are no nonzero terms in the sum and $y(n) = 0$. On the other hand, if $n \geq 0$,

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

Therefore,

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

Table 1-1 Closed-form Expressions for Some Commonly Encountered Series

$\sum_{n=0}^{N-1} a^n = \frac{1-a^N}{1-a}$	$\sum_{n=0}^{\infty} a^n = \frac{1}{1-a} \quad \lvert a\rvert < 1$
$\sum_{n=0}^{N-1} na^n = \frac{(N-1)a^{N+1} - Na^N + a}{(1-a)^2}$	$\sum_{n=0}^{\infty} na^n = \frac{a}{(1-a)^2} \quad \lvert a\rvert < 1$
$\sum_{n=0}^{N-1} n = \frac{1}{2}N(N-1)$	$\sum_{n=0}^{N-1} n^2 = \frac{1}{6}N(N-1)(2N-1)$

Graphical Approach

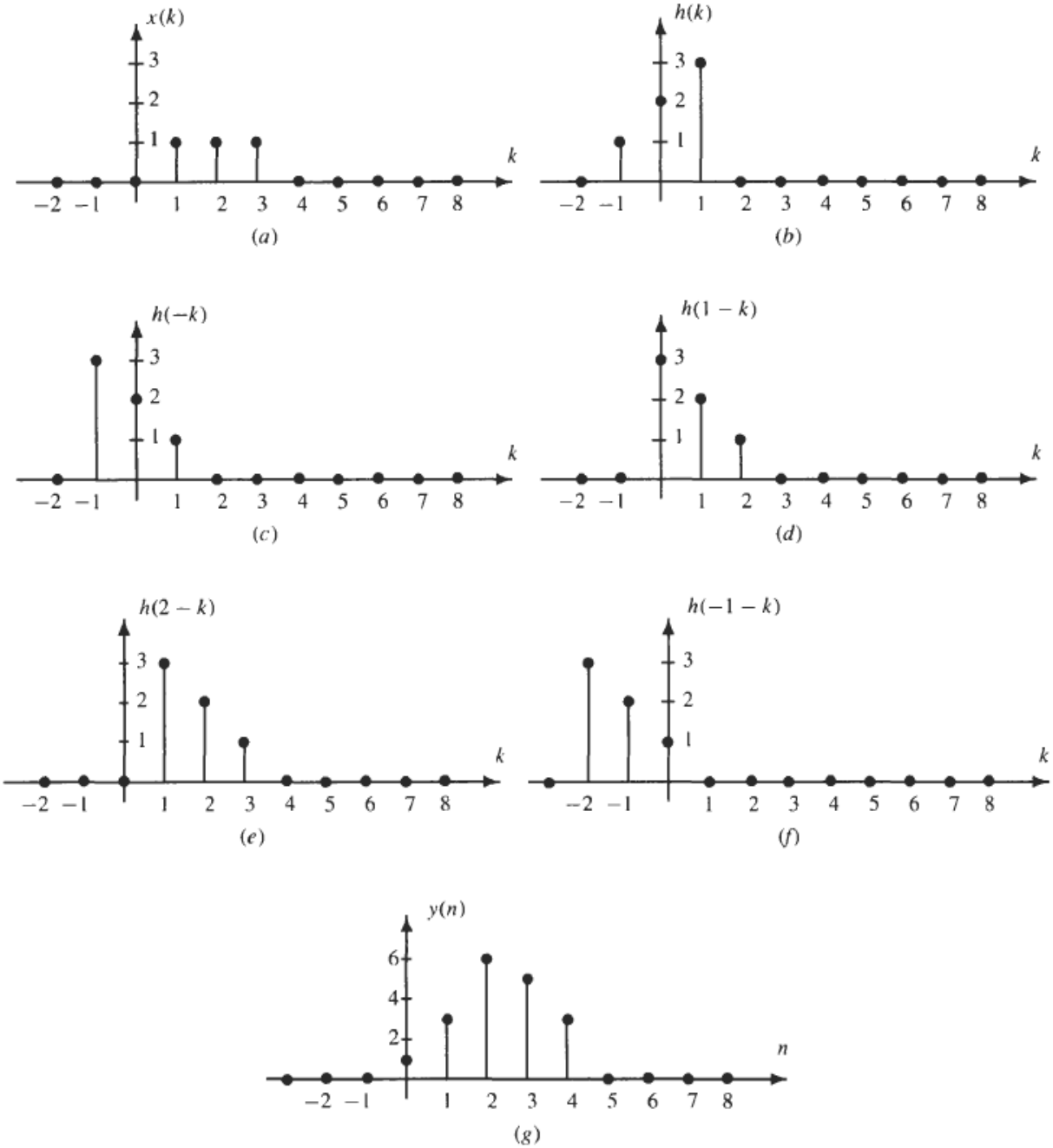
In addition to the direct method, convolutions may also be performed graphically. The steps involved in using the graphical approach are as follows:

1. Plot both sequences, $x(k)$ and $h(k)$, as functions of k.
2. Choose one of the sequences, say $h(k)$, and time-reverse it to form the sequence $h(-k)$.
3. Shift the time-reversed sequence by n. [*Note:* If $n > 0$, this corresponds to a shift to the right (delay), whereas if $n < 0$, this corresponds to a shift to the left (advance).]
4. Multiply the two sequences $x(k)$ and $h(n-k)$ and sum the product for all values of k. The resulting value will be equal to $y(n)$. This process is repeated for all possible shifts, n.

EXAMPLE 1.4.2 To illustrate the graphical approach to convolution, let us evaluate $y(n) = x(n) * h(n)$ where $x(n)$ and $h(n)$ are the sequences shown in Fig. 1-6 (*a*) and (*b*), respectively.To perform this convolution, we follow the steps listed above:

1. Because $x(k)$ and $h(k)$ are both plotted as a function of k in Fig. 1-6 (*a*) and (*b*), we next choose one of the sequences to reverse in time. In this example, we time-reverse $h(k)$, which is shown in Fig. 1-6 (*c*).
2. Forming the product, $x(k)h(-k)$, and summing over k, we find that $y(0) = 1$.
3. Shifting $h(k)$ to the right by one results in the sequence $h(1-k)$ shown in Fig. 1-6 (*d*). Forming the product, $x(k)h(1-k)$, and summing over k, we find that $y(1) = 3$.
4. Shifting $h(1-k)$ to the right again gives the sequence $h(2-k)$ shown in Fig. 1-6 (*e*). Forming the product, $x(k)h(2-k)$, and summing over k, we find that $y(2) = 6$.
5. Continuing in this manner, we find that $y(3) = 5$, $y(4) = 3$, and $y(n) = 0$ for $n > 4$.
6. We next take $h(-k)$ and shift it to the left by one as shown in Fig. 1-6 (*f*). Because the product, $x(k)h(-1-k)$, is equal to zero for all k, we find that $y(-1) = 0$. In fact, $y(n) = 0$ for all $n < 0$.

Figure 1-6 (*g*) shows the convolution for all n.



A useful fact to remember in performing the convolution of two finite-length sequences is that if $x(n)$ is of length L_1 and $h(n)$ is of length L_2, $y(n) = x(n) * h(n)$ will be of length

$$L = L_1 + L_2 - 1$$

Furthermore, if the nonzero values of $x(n)$ are contained in the interval $[M_x, N_x]$ and the nonzero values of $h(n)$ are contained in the interval $[M_h, N_h]$, the nonzero values of $y(n)$ will be *confined* to the interval $[M_x + M_h, N_x + N_h]$.

Cross-Correlation

$$r_{xy}(\ell) = \sum_{n=\infty}^{+\infty} x(n)y(n-\ell) \qquad \ell = 0, \pm 1, \pm 2$$

Definition:

$$= \sum_{n=\infty}^{+\infty} x(n+\ell)y(n)$$

$$= x(\ell) * y(-\ell)$$

$$r_{yx}(\ell) = r_{xy}(-\ell)$$

When $y(n) = x(n)$, then it is called auto-correlation.

A symmetric (even) function $r_{xx}(\ell) = \sum_{n=\infty}^{+\infty} x(n)x(n-\ell) = \sum_{n=\infty}^{+\infty} x(n)x(n+\ell) = x(n) * x(-n)$

Properties

$$r_{xx}(0) = \sum_{ns-\infty}^{+\infty} x(n) \bullet x(n) = \sum_{ns-\infty}^{+\infty} |x(n)|^2 = E_x \quad \text{energy}$$

$$|r_{xy}(\ell)| \le \sqrt{r_{xx}(0)r_{yy}(0)} = \sqrt{E_x E_y}$$

and

$$|r_{xx}(\ell)| \le r_{xx}(o) = E_x$$

Normalized auto or cross correlation:

$$\rho_{xx}(\ell) = \frac{\gamma_{xx}(\ell)}{r_{xx}(0)} \le 1 \qquad \rho_{xy}(\ell) = \frac{r_{xy}(\ell)}{\sqrt{r_{xx}(0)r_{yy}(0)}} \le 1$$

For periodic signals, the correlation function is defined in one period:

$$r_{xy}(\ell) = \lim_{M \to \infty} \frac{1}{2M+1} \sum_{n=-M}^{M} x(n)y(n-\ell),$$

where M is the number of observed samples.

If x and y are both periodic with period N, then

$$r_{xy}(\ell) = \frac{1}{N} \sum_{n=o}^{N-1} x(n)y(n-\ell) \Rightarrow \text{ correlation is also a periodic sequence with period } N.$$

1.5 DIFFERENCE EQUATIONS

The convolution sum expresses the output of a linear shift-invariant system in terms of a linear combination of the input values $x(n)$. For example, a system that has a unit sample response $h(n) = \alpha^n u(n)$ is described by the equation

$$y(n) = \sum_{k=0}^{\infty} \alpha^k x(n-k) \tag{1.9}$$

Although this equation allows one to compute the output $y(n)$ for an arbitrary input $x(n)$, from a computational point of view this representation is not very efficient. In some cases it may be possible to more efficiently express the output in terms of past values of the output in addition to the current and past values of the input. The previous system, for example, may be described more concisely as follows:

$$y(n) = \alpha y(n-1) + x(n) \tag{1.10}$$

Equation (*1.10*) is a special case of what is known as a *linear constant coefficient difference equation*, or LCCDE. The general form of a LCCDE is

$$y(n) = \sum_{k=0}^{q} b(k)x(n-k) - \sum_{k=1}^{p} a(k)y(n-k) \tag{1.11}$$

where the coefficients $a(k)$ and $b(k)$ are constants that define the system. If the difference equation has one or more terms $a(k)$ that are nonzero, the difference equation is said to be *recursive*. On the other hand, if all of the coefficients $a(k)$ are equal to zero, the difference equation is said to be *nonrecursive*. Thus, Eq. (*1.10*) is an example of a first-order recursive difference equation, whereas Eq. (*1.9*) is an infinite-order nonrecursive difference equation.

Difference equations provide a method for computing the response of a system, $y(n)$, to an arbitrary input $x(n)$. Before these equations may be solved, however, it is necessary to specify a set of *initial conditions*. For example, with an input $x(n)$ that begins at time $n = 0$, the solution to Eq. (*1.11*) at time $n = 0$ depends on the values of $y(-1), \ldots, y(-p)$. Therefore, these initial conditions must be specified before the solution for $n \geq 0$ may be found. When these initial conditions are zero, the system is said to be in *initial rest*.

Given an LCCDE, the general solution is a sum of two parts,

$$y(n) = y_h(n) + y_p(n)$$

where $y_h(n)$ is known as the *homogeneous solution* and $y_p(n)$ is the *particular solution*. The homogeneous solution is the response of the system to the initial conditions, assuming that the input $x(n) = 0$. The particular solution is the response of the system to the input $x(n)$, assuming zero initial conditions.

The homogeneous solution is found by solving the homogeneous difference equation

$$y(n) + \sum_{k=1}^{p} a(k)y(n-k) = 0 \tag{1.13}$$

The solution to Eq. (*1.13*) may be found by assuming a solution of the form

$$y_h(n) = z^n$$

Substituting this solution into Eq. (*1.13*) we obtain the polynomial equation

$$z^n + \sum_{k=1}^{p} a(k) z^{n-k} = 0$$

or

$$z^{n-p}\{z^p + a(1)\,z^{p-1} + a(2)\,z^{p-2} + \cdots + a(p-1)\,z + a(p)\} = 0$$

The polynomial in braces is called the *characteristic polynomial*. Because it is of degree p, it will have p roots, which may be either real or complex. If the coefficients $a(k)$ are real-valued, these roots will occur in complex-conjugate pairs (i.e., for each complex root z_i there will be another that is equal to z_i^*). If the p roots z_i are distinct, $z_i \neq z_k$ for $k \neq i$, the general solution to the homogeneous difference equation is

$$y_h(n) = \sum_{k=1}^{p} A_k z_k^n \tag{1.14}$$

where the constants A_k are chosen to satisfy the initial conditions. For repeated roots, the solution must be modified as follows. If z_1 is a root of multiplicity m with the remaining $p - m$ roots distinct, the homogeneous solution becomes

$$y_h(n) = (A_1 + A_2 n + \cdots + A_m n^{m-1}) z_1^n + \sum_{k=m+1}^{p} A_k z_k^n \tag{1.15}$$

For the particular solution, it is necessary to find the sequence $y_p(n)$ that satisfies the difference equation for the given $x(n)$. In general, this requires some creativity and insight. However, for many of the typical inputs that we are interested in, the solution will have the same form as the input. Table 1-2 lists the particular solution for some commonly encountered inputs. For example, if $x(n) = a^n u(n)$, the particular solution will be of the form

$$y_p(n) = C a^n u(n)$$

provided a is not a root of the characteristic equation. The constant C is found by substituting the solution into the difference equation. Note that for $x(n) = C\delta(n)$ the particular solution is zero. Because $x(n) = 0$ for $n > 0$, the unit sample only affects the initial condition of $y(n)$.

Table 1-2 The Particular Solution to an LCCDE for Several Different Inputs

Term in $x(n)$	Particular Solution
C	C_1
Cn	$C_1 n + C_2$
Ca^n	$C_1 a^n$
$C\cos(n\omega_0)$	$C_1\cos(n\omega_0) + C_2\sin(n\omega_0)$
$C\sin(n\omega_0)$	$C_1\cos(n\omega_0) + C_2\sin(n\omega_0)$
$Ca^n\cos(n\omega_0)$	$C_1 a^n\cos(n\omega_0) + C_2 a^n\sin(n\omega_0)$
$C\delta(n)$	None

EXAMPLE 1.5.1 Let us find the solution to the difference equation

$$y(n) - 0.25y(n-2) = x(n) \tag{1.16}$$

for $x(n) = u(n)$ assuming initial conditions of $y(-1) = 1$ and $y(-2) = 0$.

We begin by finding the particular solution. From Table 1-2 we see that for $x(n) = u(n)$

$$y_p(n) = C_1$$

Substituting this solution into the difference equation we find

$$C_1 - 0.25C_1 = 1$$

In order for this to hold, we must have

$$C_1 = \frac{1}{1 - 0.25} = \frac{4}{3}$$

To find the homogeneous solution, we set $y_h(n) = z^n$, which gives the characteristic polynomial

$$z^2 - 0.25 = 0$$

or

$$(z + 0.5)(z - 0.5) = 0$$

Therefore, the homogeneous solution has the form

$$y_h(n) = A_1(0.5)^n + A_2(-0.5)^n$$

Thus, the total solution is

$$y(n) = \tfrac{4}{3} + A_1(0.5)^n + A_2(-0.5)^n \qquad n \geq 0 \tag{1.17}$$

The constants A_1 and A_2 must now be found so that the total solution satisfies the given initial conditions, $y(-1) = 1$ and $y(-2) = 0$. Because the solution given in Eq. (*1.17*) only applies for $n \geq 0$, we must derive an equivalent set of initial conditions for $y(0)$ and $y(1)$. Evaluating Eq. (*1.16*) at $n = 0$ and $n = 1$, we have

$$y(0) - 0.25y(-2) = x(0) = 1$$
$$y(1) - 0.25y(-1) = x(1) = 1$$

Substituting these *derived initial conditions* into Eq. (*1.17*) we have

$$y(0) = \tfrac{4}{3} + A_1 + A_2 = 1$$
$$y(1) = \tfrac{4}{3} + \tfrac{1}{2}A_1 - \tfrac{1}{2}A_2 = 1$$

Solving for A_1 and A_2 we find

$$A_1 = -\tfrac{1}{2} \qquad A_2 = \tfrac{1}{6}$$

Thus, the solution is

$$y(n) = \tfrac{4}{3} - (0.5)^{n+1} + \tfrac{1}{6}(-0.5)^n \qquad n \geq 0$$

Although we have focused thus far on linear difference equations with constant coefficients, not all systems and not all difference equations of interest are linear, and not all have constant coefficients. A system that computes a running average of a signal $x(n)$ over the interval $[0, n]$, for example, is defined by

$$y(n) = \frac{1}{n+1}\sum_{k=0}^{n} x(k) \qquad n \geq 0$$

This system may be represented by a difference equation that has time-varying coefficients:

$$y(n) = \frac{n}{n+1}y(n-1) + x(n) \qquad n \geq 0$$

Although more complicated and difficult to solve, nonlinear difference equations or difference equations with time-varying coefficients are important and arise frequently in many applications.

Z Transform:

The z-transform is a useful tool in the analysis of discrete-time signals and systems and is the discrete-time counterpart of the Laplace transform for continuous-time signals and systems. The z-transform may be used to solve constant coefficient difference equations, evaluate the response of a linear time-invariant system to a given input, and design linear filters.

Definition: The z-transform of a discrete-time signal $x(n)$ is defined by[1]

$$X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n}$$

where $z = re^{j\omega}$ is a complex variable. The values of z for which the sum converges define a region in the z-plane referred to as the *region of convergence* (ROC).

Notationally, if $x(n)$ has a z-transform $X(z)$, we write

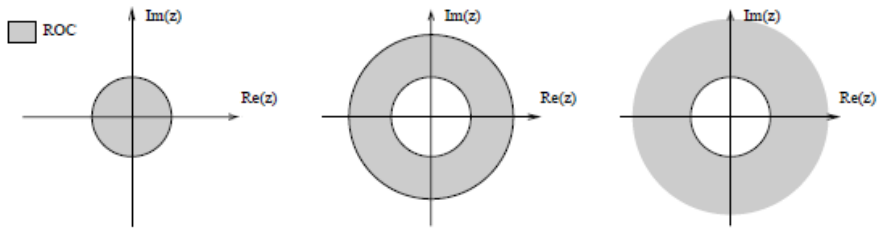
$$x(n) \overset{Z}{\longleftrightarrow} X(z)$$

The Region of Convergence (ROC) is the set of all values of *z*, where *x(z)* attains a finite value

■ 1.6.3 Properties of ROC, Poles and Zeros

The following is a list of several important properties of the z-transform.

1. The ROC is a ring or a disc centered at the origin: $r_1 < z < r_2$. Note that r_1 or r_2 could be 0 or ∞.



(a) $x(n)$ left-sided. (b) Generic $x(n)$. (c) $x(n)$ right-sided.

Figure 1.76. The geometries of the ROC.

2. The ROC cannot contain any poles.

3. If $x(n)$ is a finite-duration sequence (i.e., $x(n) = 0$ except for $N_1 \leq n \leq N_2$), then the ROC is the whole z-plane except possibly $z = 0$.

4. If $x(n)$ is a right-sided sequence ($x(n) = 0$ for $n < N_1$), then the ROC is

$$|z| > |p_{max}|,$$

where p_{max} is the outermost finite pole of $X(z)$.

5. If $x(n)$ is a left-sided sequence ($x(n) = 0$ for $n \geq N_2$), then the ROC is

$$|z| < |p_{min}|,$$

where p_{min} is the innermost non-zero pole of $X(z)$.

6. Generalizing Example 1.32 and Example 1.33, we have:

$$a^n u(n) \leftrightarrow \frac{1}{1 - az^{-1}}, \quad \text{where the ROC is } |z| > |a|, \tag{1.51}$$

$$-a^n u(-n-1) \leftrightarrow \frac{1}{1 - az^{-1}}, \quad \text{where the ROC is } |z| < |a|. \tag{1.52}$$

7. An *LTI system* is BIBO stable if and only if the ROC of its transfer function $H(z)$ includes $z = 1$, (i.e., includes the unit circle):

$$\left[\sum_n \left|h(n)z^{-n}\right|\right]_{z=1} < \infty$$

$$\Updownarrow$$

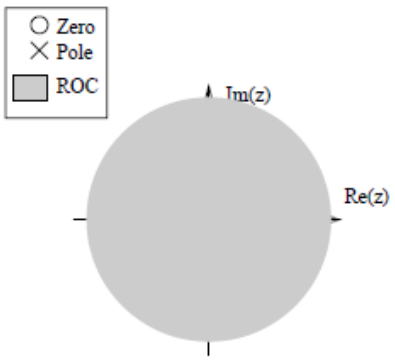
$$\sum_n |h(n)| < \infty \quad \Leftrightarrow \quad \text{BIBO stability.}$$

Example1:

Let us consider the z-transform of $a^n u(n)$*:*

$$\begin{aligned} ZT\left\{a^n u(n)\right\} &= \sum_{n=0}^{\infty} a^n z^{-n} \\ &= \sum_{n=0}^{\infty} \left(az^{-1}\right)^n \\ &= \frac{1}{1-az^{-1}} \quad \textit{if } \left|az^{-1}\right| < 1, \\ &= \frac{z}{z-a} \end{aligned}$$

The region of convergence (ROC) *is the set of all values of z for which the z-transform converges. In this example, it is* $|z| > |a|$.



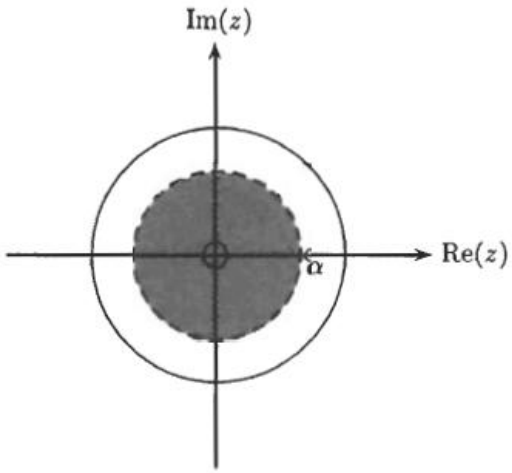
Example2:

Let us find the z-transform of the sequence $x(n) = -\alpha^n u(-n-1)$.

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

$$= -\alpha^{-1}z \sum_{n=0}^{\infty} (\alpha^{-1}z)^n = -\frac{\alpha^{-1}z}{1-\alpha^{-1}z} = \frac{1}{1-\alpha z^{-1}}$$

with the sum converging if $|\alpha^{-1}z| < 1$ or $|z| < |\alpha|$. A pole-zero diagram with the region of convergence indicated is given in the figure below.



Common z-Transform Pairs

Sequence	z-Transform	Region of Convergence
$\delta(n)$	1	all z
$\alpha^n u(n)$	$\dfrac{1}{1-\alpha z^{-1}}$	$\lvert z\rvert > \lvert\alpha\rvert$
$-\alpha^n u(-n-1)$	$\dfrac{1}{1-\alpha z^{-1}}$	$\lvert z\rvert < \lvert\alpha\rvert$
$n\alpha^n u(n)$	$\dfrac{\alpha z^{-1}}{(1-\alpha z^{-1})^2}$	$\lvert z\rvert > \lvert\alpha\rvert$
$-n\alpha^n u(-n-1)$	$\dfrac{\alpha z^{-1}}{(1-\alpha z^{-1})^2}$	$\lvert z\rvert < \lvert\alpha\rvert$
$\cos(n\omega_0)u(n)$	$\dfrac{1-(\cos\omega_0)z^{-1}}{1-2(\cos\omega_0)z^{-1}+z^{-2}}$	$\lvert z\rvert > 1$
$\sin(n\omega_0)u(n)$	$\dfrac{(\sin\omega_0)z^{-1}}{1-2(\cos\omega_0)z^{-1}+z^{-2}}$	$\lvert z\rvert > 1$

Properties:

Linearity

As with the DTFT, the z-transform is a *linear* operator. Therefore, if $x(n)$ has a z-transform $X(z)$ with a region of convergence R_x, and if $y(n)$ has a z-transform $Y(z)$ with a region of convergence R_y,

$$w(n) = ax(n) + by(n) \overset{Z}{\longleftrightarrow} W(z) = aX(z) + bY(z)$$

and the ROC of $w(n)$ will *include* the intersection of R_x and R_y, that is,

$$R_w \text{ contains } R_x \cap R_y$$

However, the region of convergence of $W(z)$ *may* be larger. For example, if $x(n) = u(n)$ and $y(n) = u(n-1)$, the ROC of $X(z)$ and $Y(z)$ is $|z| > 1$. However, the z-transform of $w(n) = x(n) - y(n) = \delta(n)$ is the entire z-plane.

Time Reversal

If $x(n)$ has a z-transform $X(z)$ with a region of convergence R_x that is the annulus $\alpha < |z| < \beta$, the z-transform of the time-reversed sequence $x(-n)$ is

$$x(-n) \overset{Z}{\longleftrightarrow} X(z^{-1})$$

and has a region of convergence $1/\beta < |z| < 1/\alpha$, which is denoted by $1/R_x$.

Shifting Property

Shifting a sequence (delaying or advancing) multiplies the z-transform by a power of z. That is to say, if $x(n)$ has a z-transform $X(z)$,

$$x(n - n_0) \overset{Z}{\longleftrightarrow} z^{-n_0} X(z)$$

Because shifting a sequence does not affect its absolute summability, shifting does not change the region of convergence. Therefore, the z-transforms of $x(n)$ and $x(n - n_0)$ have the same region of convergence, with the possible exception of adding or deleting the points $z = 0$ and $z = \infty$.

Multiplication by an Exponential

If a sequence $x(n)$ is multiplied by a complex exponential α^n,

$$\alpha^n x(n) \overset{Z}{\longleftrightarrow} X(\alpha^{-1} z)$$

This corresponds to a scaling of the z-plane. If the region of convergence of $X(z)$ is $r_- < |z| < r_+$, which will be denoted by R_x, the region of convergence of $X(\alpha^{-1}z)$ is $|\alpha| r_- < |z| < |\alpha| r_+$, which is denoted by $|\alpha| R_x$. As a special case, note that if $x(n)$ is multiplied by a complex exponential, $e^{jn\omega_0}$,

$$e^{jn\omega_0} x(n) \overset{Z}{\longleftrightarrow} X(e^{-j\omega_0} z)$$

which corresponds to a rotation of the z-plane.

Convolution Theorem

Perhaps the most important z-transform property is the convolution theorem, which states that convolution in the time domain is mapped into multiplication in the frequency domain, that is,

$$y(n) = x(n) * h(n) \overset{Z}{\longleftrightarrow} Y(z) = X(z)H(z)$$

The region of convergence of $Y(z)$ includes the intersection of R_x and R_y,

$$R_w \text{ contains } R_x \cap R_y$$

However, the region of convergence of $Y(z)$ *may* be larger, if there is a pole-zero cancellation in the product $X(z)H(z)$.

Derivative

If $X(z)$ is the z-transform of $x(n)$, the z-transform of $nx(n)$ is

$$nx(n) \overset{Z}{\longleftrightarrow} -z \frac{dX(z)}{dz}$$

Repeated application of this property allows for the evaluation of the z-transform of $n^k x(n)$ for any integer k.

These properties are summarized in Table 4-2. As illustrated in the following example, these properties are useful in simplifying the evaluation of z-transforms.

Conjugation

If $X(z)$ is the z-transform of $x(n)$, the z-transform of the complex conjugate of $x(n)$ is

$$x^*(n) \overset{Z}{\longleftrightarrow} X^*(z^*)$$

As a corollary, note that if $x(n)$ is real-valued, $x(n) = x^*(n)$, then

$$X(z) = X^*(z^*)$$

Initial Value Theorem

If $x(n)$ is equal to zero for $n < 0$, the initial value, $x(0)$, may be found from $X(z)$ as follows:

$$x(0) = \lim_{z \to \infty} X(z)$$

This property is a consequence of the fact that if $x(n) = 0$ for $n < 0$,

$$X(z) = x(0) + x(1)\,z^{-1} + x(2)\,z^{-2} + \cdots$$

Therefore, if we let $z \to \infty$, each term in $X(z)$ goes to zero except the first.

Example:

find the z-transform of $x(n) = n\alpha^n u(-n)$. To find $X(z)$, we will use the time-reversal and derivative properties.

$$\alpha^n u(n) \overset{Z}{\longleftrightarrow} \frac{1}{1 - \alpha z^{-1}} \qquad |z| > \alpha$$

Therefore,

$$\left(\frac{1}{\alpha}\right)^n u(n) \overset{Z}{\longleftrightarrow} \frac{1}{1 - \alpha^{-1} z^{-1}} \qquad |z| > \frac{1}{\alpha}$$

and, using the time-reversal property,

$$\alpha^n u(-n) \overset{Z}{\longleftrightarrow} \frac{1}{1 - \alpha^{-1} z} \qquad |z| < \alpha$$

Finally, using the derivative property, it follows that the z-transform of $n\alpha^n u(-n)$ is

$$-z \frac{d}{dz} \frac{1}{1 - \alpha^{-1} z} = -\frac{\alpha^{-1} z}{(1 - \alpha^{-1} z)^2} \qquad |z| < \alpha$$

A property that may be used to find the initial value of a causal sequence from its z-transform is the initial value theorem.

Final Value Theorem. Suppose that $x(k)$, where $x(k) = 0$ for $k < 0$, has the z transform $X(z)$ and that all the poles of $X(z)$ lie inside the unit circle, with the possible exception of a simple pole at $z = 1$. [This is the condition for the stability of $X(z)$, or the condition for $x(k)$ ($k = 0,1,2, \ldots$) to remain finite.] Then the final value of x(k), that is, the value of x(k) as k approaches infinity, can be given by

$$\lim_{k\to\infty} x(k) = \lim_{z\to 1}[(1 - z^{-1})X(z)]$$

To prove the final value theorem, note that

$$Z[x(k)] = X(z) = \sum_{k=0}^{\infty} x(k)z^{-k}$$

$$Z[x(k-1)] = z^{-1}X(z) = \sum_{k=0}^{\infty} x(k-1)z^{-k}$$

Hence,

$$\sum_{k=0}^{\infty} x(k)z^{-k} - \sum_{k=0}^{\infty} x(k-1)z^{-k} = X(z) - z^{-1}X(z)$$

Taking the limit as z approaches unity, we have

$$\lim_{z\to 1}\left[\sum_{k=0}^{\infty} x(k)z^{-k} - \sum_{k=0}^{\infty} x(k-1)z^{-k}\right] = \lim_{z\to 1}[(1 - z^{-1})X(z)]$$

Because of the assumed stability condition and the condition that $x(k) = 0$ for $k < 0$, the left-hand side of this last equation becomes

$$\sum_{k=0}^{\infty}[x(k) - x(k-1)] = [x(0) - x(-1)] + [x(1) - x(0)] + [x(2) - x(1)] + \cdots = x(\infty) = \lim_{k\to\infty} x(k)$$

Hence,

$$\lim_{k\to\infty} x(k) = \lim_{z\to 1}[(1 - z^{-1})X(z)]$$

The final value theorem is very useful in determining the behavior of $x(k)$ as $k \to \infty$ from its z transform $X(z)$.

THE INVERSE z-TRANSFORM

The z-transform is a useful tool in linear systems analysis. However, just as important as techniques for finding the z-transform of a sequence are methods that may be used to invert the z-transform and recover the sequence $x(n)$ from $X(z)$. Three possible approaches are described below.

1 *Partial Fraction Expansion*

For z-transforms that are rational functions of z,

$$X(z) = \frac{\sum_{k=0}^{q} b(k)z^{-k}}{\sum_{k=0}^{p} a(k)z^{-k}} = C\,\frac{\prod_{k=1}^{q}(1-\beta_k z^{-1})}{\prod_{k=1}^{p}(1-\alpha_k z^{-1})}$$

a simple and straightforward approach to find the inverse z-transform is to perform a partial fraction expansion of $X(z)$. Assuming that $p > q$, and that all of the roots in the denominator are simple, $\alpha_i \neq \alpha_k$ for $i \neq k$, $X(z)$ may be expanded as follows:

$$X(z) = \sum_{k=1}^{p} \frac{A_k}{1-\alpha_k z^{-1}} \tag{4.5}$$

for some constants A_k for $k = 1, 2, \ldots, p$. The coefficients A_k may be found by multiplying both sides of Eq. (4.5) by $(1-\alpha_k z^{-1})$ and setting $z = \alpha_k$. The result is

$$A_k = \left[(1-\alpha_k z^{-1})X(z)\right]_{z=\alpha_k}$$

If $p \leq q$, the partial fraction expansion must include a polynomial in z^{-1} of order $(p-q)$. The coefficients of this polynomial may be found by long division (i.e., by dividing the numerator polynomial by the denominator). For multiple-order poles, the expansion must be modified. For example, if $X(z)$ has a second-order pole at $z = \alpha_k$, the expansion will include two terms,

$$\frac{B_1}{1-\alpha_k z^{-1}} + \frac{B_2}{(1-\alpha_k z^{-1})^2}$$

where B_1 and B_2 are given by

$$B_1 = \alpha_k\left[\frac{d}{dz}(1-\alpha_k z^{-1})^2 X(z)\right]_{z=\alpha_k}$$

$$B_2 = \left[(1-\alpha_k z^{-1})^2 X(z)\right]_{z=\alpha_k}$$

EXAMPLE Suppose that a sequence $x(n)$ has a z-transform

$$X(z) = \frac{4 - \frac{7}{4}z^{-1} + \frac{1}{4}z^{-2}}{1 - \frac{3}{4}z^{-1} + \frac{1}{8}z^{-2}} = \frac{4 - \frac{7}{4}z^{-1} + \frac{1}{4}z^{-2}}{\left(1-\frac{1}{2}z^{-1}\right)\left(1-\frac{1}{4}z^{-1}\right)}$$

with a region of convergence $|z| > \frac{1}{2}$. Because $p = q = 2$, and the two poles are simple, the partial fraction expansion has the form

$$X(z) = C + \frac{A_1}{1-\frac{1}{2}z^{-1}} + \frac{A_2}{1-\frac{1}{4}z^{-1}}$$

The constant C is found by long division:

$$\begin{array}{r|l} & 2 \\ \hline \frac{1}{8}z^{-2} - \frac{3}{4}z^{-1} + 1 & \frac{1}{4}z^{-2} - \frac{7}{4}z^{-1} + 4 \\ & \underline{\frac{1}{4}z^{-2} - \frac{3}{2}z^{-1} + 2} \\ & -\frac{1}{4}z^{-1} + 2 \end{array}$$

Therefore, $C = 2$ and we may write $X(z)$ as follows:

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

Next, for the coefficients A_1 and A_2 we have

$$A_1 = \left[\left(1 - \tfrac{1}{2}z^{-1}\right)X(z)\right]_{z^{-1}=2} = \left.\frac{4 - \frac{7}{4}z^{-1} + \frac{1}{4}z^{-2}}{1 - \frac{1}{4}z^{-1}}\right|_{z^{-1}=2} = 3$$

and

$$A_2 = \left[\left(1 - \tfrac{1}{4}z^{-1}\right)X(z)\right]_{z^{-1}=4} = \left.\frac{4 - \frac{7}{4}z^{-1} + \frac{1}{4}z^{-2}}{1 - \frac{1}{2}z^{-1}}\right|_{z^{-1}=4} = -1$$

Thus, the complete partial fraction expansion becomes

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

Finally, because the region of convergence is the exterior of the circle $|z| > \frac{1}{2}$, $x(n)$ is the right-sided sequence

$$x(n) = 2\delta(n) + 3\left(\tfrac{1}{2}\right)^n u(n) - \left(\tfrac{1}{4}\right)^n u(n)$$

2 Power Series

The z-transform is a power series expansion,

$$X(z) = \sum_{n=-\infty}^{\infty} x(n)z^{-n} = \cdots + x(-2)\,z^2 + x(-1)\,z + x(0) + x(1)\,z^{-1} + x(2)\,z^{-2} + \cdots$$

where the sequence values $x(n)$ are the coefficients of z^{-n} in the expansion. Therefore, if we can find the power series expansion for $X(z)$, the sequence values $x(n)$ may be found by simply picking off the coefficients of z^{-n}.

EXAMPLE Consider the z-transform

$$X(z) = \log(1 + az^{-1}) \qquad |z| > |a|$$

The power series expansion of this function is

$$\log(1 + az^{-1}) = \sum_{n=1}^{\infty} \frac{1}{n}(-1)^{n+1} a^n z^{-n}$$

Therefore, the sequence $x(n)$ having this z-transform is

$$x(n) = \begin{cases} \frac{1}{n}(-1)^{n+1} a^n & n > 0 \\ 0 & n \leq 0 \end{cases}$$

Difference Equation *To see how the z-transform is related to the time-domain representation, we expand it :*

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

Inverting the z-transform, we have:

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

EXAMPLE Consider the linear constant coefficient difference equation

$$y(n) = 0.25y(n-2) + x(n)$$

Let us find the solution to this equation assuming that $x(n) = \delta(n-1)$ with $y(-1) = y(-2) = 1$.

We begin by noting that if the one-sided z-transform of $y(n)$ is $Y_1(z)$, the one-sided z-transform of $y(n-2)$ is

$$\sum_{n=0}^{\infty} y(n-2)z^{-n} = y(-2) + y(-1)z^{-1} + \sum_{n=0}^{\infty} y(n)z^{-n-2} = y(-2) + y(-1)z^{-1} + z^{-2}Y_1(z)$$

Therefore, taking the z-transform of both sides of the difference equation, we have

$$Y_1(z) = 0.25[y(-2) + y(-1)z^{-1} + z^{-2}Y_1(z)] + X_1(z)$$

where $X_1(z) = z^{-1}$. Substituting for $y(-1)$ and $y(-2)$, and solving for $Y_1(z)$, we have

$$Y_1(z) = \tfrac{1}{4}\frac{1 + 5z^{-1}}{1 - \frac{1}{4}z^{-2}}$$

To find $y(n)$, note that $Y_1(z)$ may be expanded as follows:[2]

$$Y_1(z) = \frac{\frac{11}{8}}{1 - \frac{1}{2}z^{-1}} - \frac{\frac{9}{8}}{1 + \frac{1}{2}z^{-1}}$$

Therefore,

$$y(n) = \left[\tfrac{11}{8}\left(\tfrac{1}{2}\right)^n - \tfrac{9}{8}\left(-\tfrac{1}{2}\right)^n\right]u(n)$$

Example2:

Evaluate the convolution of the two sequences

$$h(n) = (0.5)^n u(n) \quad \text{and} \quad x(n) = 3^n u(-n)$$

To evaluate this convolution, we will use the convolution property of the z-transform. The z-transform of $h(n)$ is

$$H(z) = \frac{1}{1 - \frac{1}{2}z^{-1}} \qquad |z| > \tfrac{1}{2}$$

and the z-transform of $x(n)$ may be found from the time-reversal and shift properties, or directly as follows:

$$\begin{aligned} X(z) &= \sum_{n=-\infty}^{\infty} x(n)z^{-n} = \sum_{n=-\infty}^{0} 3^n z^{-n} \\ &= \sum_{n=0}^{\infty}\left(\tfrac{1}{3}z\right)^n = \frac{1}{1 - \frac{1}{3}z} = -\frac{3z^{-1}}{1 - 3z^{-1}} \qquad |z| < 3 \end{aligned}$$

Therefore, the z-transform of the convolution, $y(n) = x(n) * h(n)$, is

$$Y(z) = -\frac{1}{1 - \frac{1}{2}z^{-1}} \cdot \frac{3z^{-1}}{1 - 3z^{-1}}$$

The region of convergence is the intersection of the regions $|z| > \frac{1}{2}$ and $|z| < 3$, which is $\frac{1}{2} < |z| < 3$. To find the inverse z-transform, we perform a partial fraction expansion of $Y(z)$,

$$Y(z) = \frac{A}{1 - \frac{1}{2}z^{-1}} + \frac{B}{1 - 3z^{-1}}$$

where $$A = \left[\left(1 - \tfrac{1}{2}z^{-1}\right)Y(z)\right]_{z=\frac{1}{2}} = \tfrac{6}{5}$$

and $$B = [(1 - 3z^{-1})Y(z)]_{z=3} = -\tfrac{6}{5}$$

Therefore, it follows that

$$y(n) = \left(\tfrac{6}{5}\right)\left(\tfrac{1}{2}\right)^n u(n) + \left(\tfrac{6}{5}\right)3^n u(-n-1)$$

UNIT – II

DISCRETE FOURIER TRANSFORM:

The sequence *X(k)* is called the N-point DFT of *x(n).* These coefficients are related to *x(n)* as follows:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N} \qquad 0 \le k < N$$

Let *x(n)* be a finite-length sequence of length N that is equal to zero outside the interval [0, N - l].

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} X(k)e^{j2\pi nk/N} \qquad 0 \le n < N$$

we write DFT pair as,

$$x(n) \overset{DFT}{\Longleftrightarrow} X(k)$$

Comparison of DFT and DTFT:

Comparing the definition of the DFT of $x(n)$ to the DTFT, it follows that the DFT coefficients are *samples* of the DTFT:

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j2\pi nk/N} = \sum_{n=-\infty}^{\infty} x(n)e^{-j2\pi nk/N} = X(e^{j\omega})|_{\omega=2\pi k/N}$$

Comparison of DFT and Z transform:

the DFT coefficients correspond to N samples of $X(z)$ that are taken at N equally spaced points around the unit circle:

$$X(k) = X(z)\big|_{z=\exp[j2\pi k/N]}$$

DFT PROPERTIES

In this section, we list some of the properties of the DFT. Because each sequence is assumed to be finite in length, some care must be exercised in manipulating DFTs.

Linearity

If $x_1(n)$ and $x_2(n)$ have N-point DFTs $X_1(k)$ and $X_2(k)$, respectively,

$$ax_1(n) + bx_2(n) \overset{DFT}{\Longleftrightarrow} aX_1(k) + bX_2(k)$$

In using this property, it is important to ensure that the DFTs are the same length. If $x_1(n)$ and $x_2(n)$ have different lengths, the shorter sequence must be *padded* with zeros in order to make it the same length as the longer sequence. For example, if $x_1(n)$ is of length N_1 and $x_2(n)$ is of length N_2 with $N_2 > N_1$, $x_1(n)$ may be considered to be a sequence of length N_2 with the last $N_2 - N_1$ values equal to zero, and DFTs of length N_2 may be taken for both sequences.

Symmetry

If $x(n)$ is real-valued, $X(k)$ is *conjugate symmetric*,

$$X(k) = X^*((-k)) = X^*((N-k))_N$$

and if $x(n)$ is imaginary, $X(k)$ is *conjugate antisymmetric*,

$$X(k) = -X^*((-k)) = -X^*((N-k))_N$$

Circular Shift

The circular shift of a sequence $x(n)$ is defined as follows:

$$x((n-n_0))_N \mathcal{R}_N(n) = \tilde{x}(n-n_0)\mathcal{R}_N(n)$$

where n_0 is the amount of the shift and $\mathcal{R}_N(n)$ is a rectangular window:

$$\mathcal{R}_N(n) = \begin{cases} 1 & 0 \le n < N \\ 0 & \text{else} \end{cases}$$

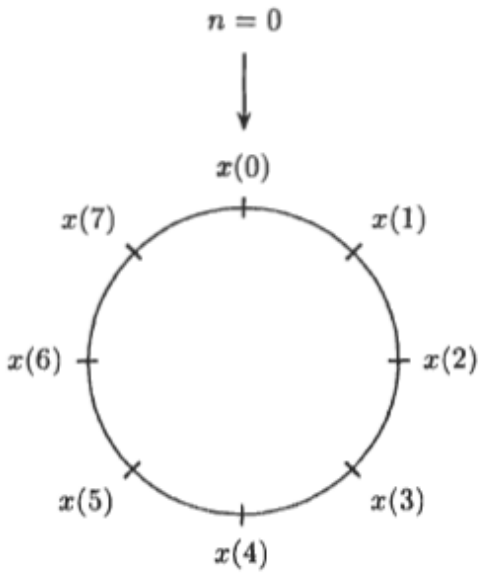
A circular shift may be visualized as follows. Suppose that the values of a sequence $x(n)$, from $n = 0$ to $n = N-1$, are marked around a circle as illustrated in Fig. or in an eight-point sequence. A circular shift to the right by n_0 corresponds to a rotation of the circle n_0 positions in a clockwise direction. An example illustrating the circular shift of a four-point sequence is shown in Fig. . Another way to circularly shift a sequence is to form the periodic sequence $\tilde{x}(n)$, perform a linear shift, $\tilde{x}(n-n_0)$, and then extract one period of $\tilde{x}(n-n_0)$ by multiplying by a rectangular window.

If a sequence is circularly shifted, the DFT is multiplied by a complex exponential,

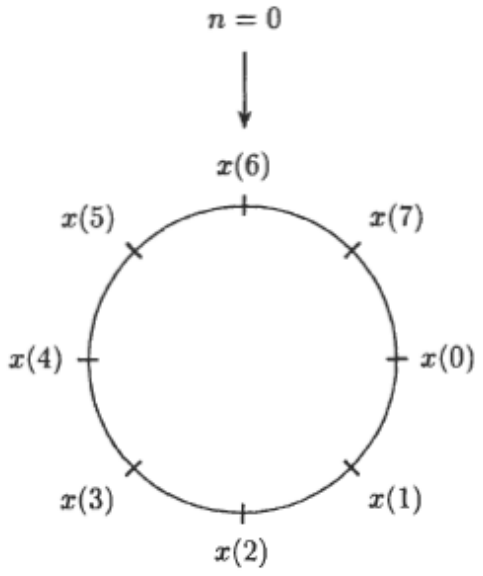
$$x((n-n_0))_N \mathcal{R}_N(n) \overset{DFT}{\Longleftrightarrow} W_N^{n_0 k} X(k)$$

Similarly, with a circular shift of the DFT, $X((k-k_0))_N$, the sequence is multiplied by a complex exponential,

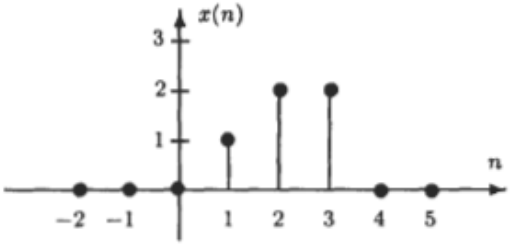
$$W_N^{nk_0} x(n) \overset{DFT}{\Longleftrightarrow} X((k+k_0))_N$$



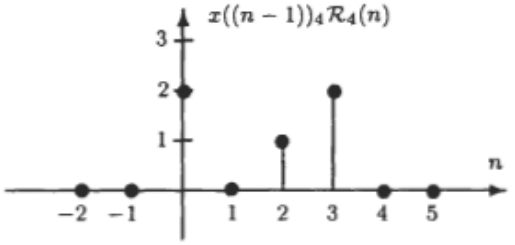
(a) An eight-point sequence.



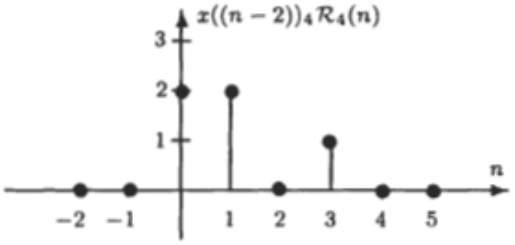
(b) Circular shift by two.



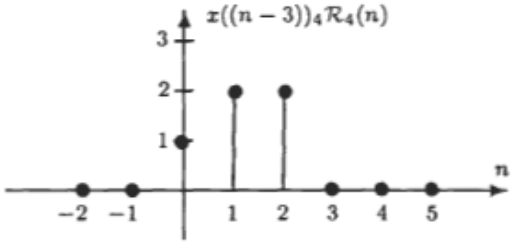
(a) A discrete-time signal of length $N = 4$.



(b) Circular shift by one.



(c) Circular shift by two.



(d) Circular shift by three.

Circular Convolution

Let $h(n)$ and $x(n)$ be finite-length sequences of length N with N-point DFTs $H(k)$ and $X(k)$, respectively. The sequence that has a DFT equal to the product $Y(k) = H(k)X(k)$ is

$$y(n) = \left[\sum_{k=0}^{N-1} \tilde{h}(k)\tilde{x}(n-k)\right]\mathcal{R}_N(n) = \left[\sum_{k=0}^{N-1} \tilde{h}(n-k)\tilde{x}(k)\right]\mathcal{R}_N(n)$$

where $\tilde{x}(n)$ and $h(n)$ are the periodic extensions of the sequences $x(n)$ and $h(n)$, respectively. Because $\tilde{h}(n) = h(n)$ for $0 \leq n < N$, the sum in Eq. may also be written as

$$y(n) = \left[\sum_{k=0}^{N-1} h(k)\tilde{x}(n-k)\right]\mathcal{R}_N(n)$$

The sequence $y(n)$ in Eq. is the *N-point circular convolution* of $h(n)$ with $x(n)$, and it is written as

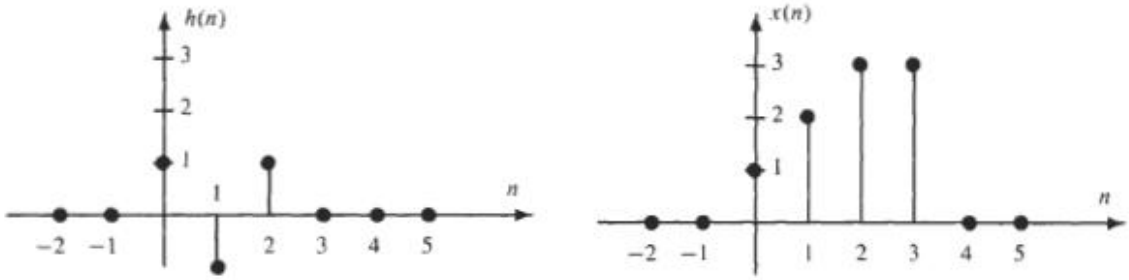
$$y(n) = h(n) \textcircled{N} x(n) = x(n) \textcircled{N} h(n)$$

The circular convolution of two finite-length sequences $h(n)$ and $x(n)$ is equivalent to one period of the periodic convolution of the periodic sequences $\tilde{h}(n)$ and $\tilde{x}(n)$,

$$y(n) = h(n) \textcircled{N} x(n) = [\tilde{h}(n) \circledast \tilde{x}(n)]\mathcal{R}_N(n)$$

In general, circular convolution is not the same as *linear* convolution, and N-point circular convolution is different, in general, from M-point circular convolution when $M \neq N$.

EXAMPLE 1 Let us perform the four-point circular convolution of the two sequences $h(n)$ and $x(n)$ shown below.



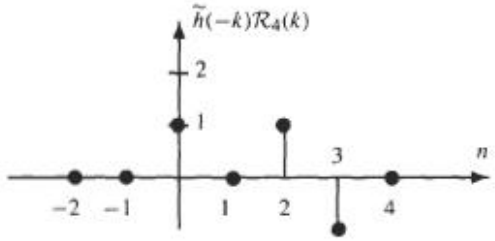
The four-point circular convolution is

$$y(n) = \left[\sum_{k=0}^{3} \tilde{h}(n-k)\tilde{x}(k)\right] \mathcal{R}_4(n)$$

which may be performed graphically, as follows. The value of $y(n)$ at $n = 0$ is

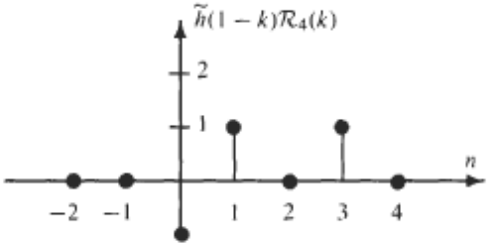
$$y(0) = \sum_{k=0}^{3} \tilde{h}(-k)\tilde{x}(k)$$

Shown in the figure below is a plot of the sequence $\tilde{h}(-k)\mathcal{R}_4(k)$.



To evaluate $y(0)$, we multiply this sequence by $x(k)$ and sum the product from $k = 0$ to $k = 3$. The result is $y(0) = 1$. Next, to find the value of $y(1)$, we evaluate the sum

Shown in the figure below is a plot of $\tilde{h}(1-k)\mathcal{R}_4(n)$.



Multiplying by $\tilde{x}(k)$ and summing from $k = 0$ to $k = 3$, we find that $y(1) = 4$. Repeating for $n = 2$ and $n = 3$, we have

$$y(2) = \sum_{k=0}^{3} \tilde{h}(2-k)\tilde{x}(k) = 2$$

$$y(3) = \sum_{k=0}^{3} \tilde{h}(3-k)\tilde{x}(k) = 2$$

Therefore, $$y(n) = h(n) \textcircled{4} x(n) = \delta(n) + 4\delta(n-1) + 2\delta(n-2) + 2\delta(n-3)$$

By comparison, the linear convolution of $h(n)$ with $x(n)$ is the following six-length sequence:

$$h(n) * x(n) = \delta(n) + \delta(n-1) + 2\delta(n-2) + 2\delta(n-3) + 3\delta(n-5)$$

Another way to perform circular convolution is to compute the DFTs of each sequence, multiply, and compute the inverse DFT.

EXAMPLE 6.4.2 Let us perform the N-point circular convolution of $x_1(n)$ and $x_2(n)$ where

$$x_1(n) = x_2(n) = \begin{cases} 1 & 0 \le n \le N-1 \\ 0 & \text{else} \end{cases}$$

Because the N-point DFTs of $x_1(n)$ and $x_2(n)$ are

$$X_1(k) = X_2(k) = \sum_{n=0}^{N-1} W_N^{nk} = \begin{cases} N & k = 0 \\ 0 & \text{else} \end{cases}$$

then $$X(k) = X_1(k)X_2(k) = \begin{cases} N^2 & k = 0 \\ 0 & \text{else} \end{cases}$$

Therefore, the N-point circular convolution of $x_1(n)$ with $x_2(n)$ is the inverse DFT of $X(k)$, which is

$$x(n) = \begin{cases} N & 0 \le n \le N-1 \\ 0 & \text{else} \end{cases}$$

Circular Versus Linear Convolution

In general, circular convolution is not the same as linear convolution. However, there is a simple relationship between circular and linear convolution that illustrates what steps must be taken in order to ensure that they are the same. Specifically, let $x(n)$ and $h(n)$ be finite-length sequences and let $y(n)$ be the linear convolution

$$y(n) = x(n) * h(n)$$

The N-point circular convolution of $x(n)$ with $h(n)$ is related to $y(n)$ as follows:

$$h(n) \circledN x(n) = \left[\sum_{k=-\infty}^{\infty} y(n + kN) \right] \mathcal{R}_N(n) \qquad (\ .1\ )$$

In other words, the circular convolution of two sequences is found by performing the linear convolution and *aliasing* the result.

An important property that follows from Eq. (.1) is that if $y(n)$ is of length N or less, $y(n - kN)$ $\mathcal{R}_N(n) = 0$ for $k \neq 0$ and

$$h(n) \circledN x(n) = h(n) * x(n)$$

that is, circular convolution is equivalent to linear convolution. Thus, if $h(n)$ and $x(n)$ are finite-length sequences of length N_1 and N_2, respectively, $y(n) = h(n) * x(n)$ is of length $N_1 + N_2 - 1$, and the N-point circular convolution is equivalent to linear convolution provided $N \geq N_1 + N_2 - 1$.

LINEAR CONVOLUTION USING THE DFT

The DFT provides a convenient way to perform convolutions without having to evaluate the convolution sum. Specifically, if $h(n)$ is N_1 points long and $x(n)$ is N_2 points long, $h(n)$ may be linearly convolved with $x(n)$ as follows:

1. Pad the sequences $h(n)$ and $x(n)$ with zeros so that they are of length $N \geq N_1 + N_2 - 1$.
2. Find the N-point DFTs of $h(n)$ and $x(n)$.
3. Multiply the DFTs to form the product $Y(k) = H(k)X(k)$.
4. Find the inverse DFT of $Y(k)$.

Sectioned Convolution:

In spite of its computational advantages, there are some difficulties with the DFT approach. For example, if *x(n)* is *very long,* we must commit a significant amount of time computing very long DFTs and in the process accept very long processing delays. In some cases, it may even be possible that *x(n)* is *roo long* to compute the DFT. The solution to these problems is to use *block convolution,* which involves segmenting the signal to be filtered, *x(n),* into sections. Each section is then filtered with the FIR filter *h(n),* and the filtered sections are pieced together to form the sequence *y(n).* There are two block convolution techniques. The first is overlap-add, and the second is overlap-save.

Overlap-Add

Let $x(n)$ be a sequence that is to be convolved with a causal FIR filter $h(n)$ of length L:

$$y(n) = h(n) * x(n) = \sum_{k=0}^{L-1} h(k)x(n-k)$$

Assume that $x(n) = 0$ for $n < 0$ and that the length of $x(n)$ is much greater than L. In the overlap-add method, $x(n)$ is partitioned into nonoverlapping subsequences of length M as illustrated in Fig. Thus, $x(n)$ may be written as a sum of shifted finite-length sequences of length M,

$$x(n) = \sum_{i=0}^{\infty} x_i(n - Mi)$$

where $$x_i(n) = \begin{cases} x(n+Mi) & n = 0, 1, \ldots, M-1 \\ 0 & \text{else} \end{cases}$$

Therefore, the linear convolution of $x(n)$ with $h(n)$ is

$$y(n) = x(n) * h(n) = \sum_{i=0}^{\infty} x_i(n - Mi) * h(n) = \sum_{i=0}^{\infty} y_i(n - Mi) \qquad (1)$$

where $y_i(n)$ is the linear convolution of $x_i(n)$ with $h(n)$,

$$y_i(n) = x_i(n) * h(n)$$

Overlap-Save

The second way that the DFT may be used to perform linear convolution is to use the *overlap-save method*. This method takes advantage of the fact that the aliasing that occurs in circular convolution only affects a portion of the sequence. For example, if $x(n)$ and $h(n)$ are finite-length sequences of lengths L and N, respectively, the linear convolution $y(n)$ is a finite-length sequence of lengths $N + L - 1$. Therefore, assuming that $N > L$, if we perform an N-point circular convolution of $x(n)$ with $h(n)$,

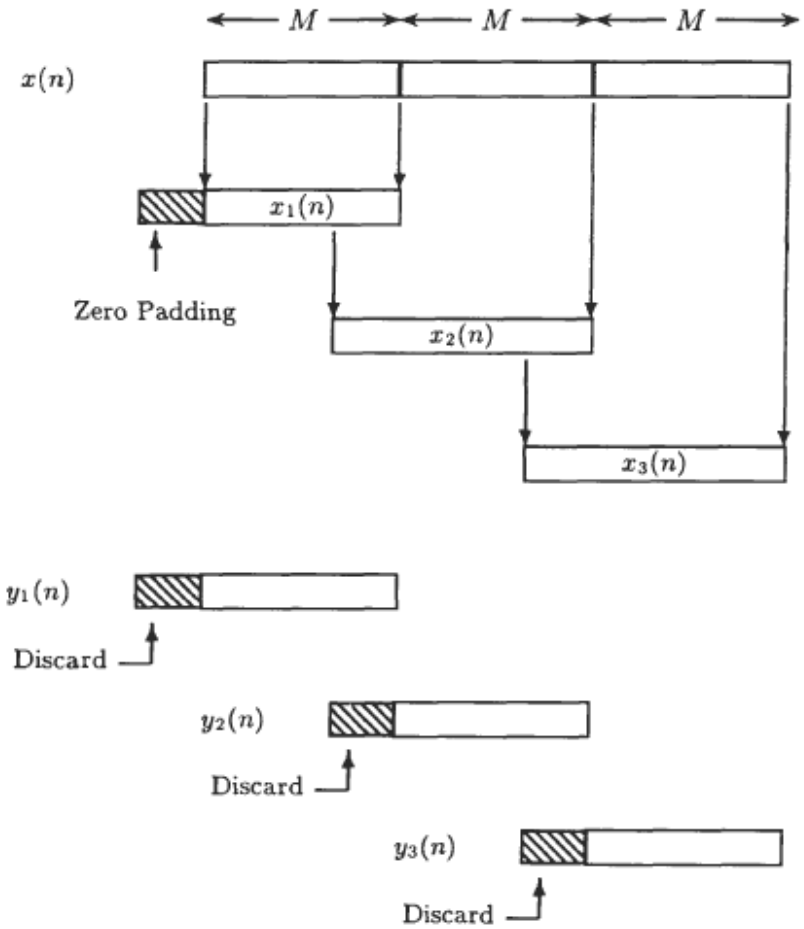
$$h(n) \circledN x(n) = \left[\sum_{k=-\infty}^{\infty} y(n + kN) \right] \mathcal{R}_N(n)$$

Because $y(n + N)$ is the only term that is aliased into the interval $0 \le n \le N - 1$, and because $y(n + N)$ only overlaps the first $L - 1$ values of $y(n)$, the remaining values in the circular convolution will not be aliased. In other words, the first $L - 1$ values of the circular convolution are not equal to the linear convolution, whereas the last $M = N - L + 1$ values are the same (see Fig.). Thus, with the appropriate *partitioning* of the input sequence $x(n)$, linear convolution may be performed by *piecing together* circular convolutions. The procedure is as follows:

1. Let $x_1(n)$ be the sequence

$$x_1(n) = \begin{cases} 0 & 0 \le n < L-1 \\ x(n-L+1) & L-1 \le n \le N-1 \end{cases}$$

2. Perform the N-point circular convolution of $x_1(n)$ with $h(n)$ by forming the product $H(k)X_1(k)$ and then finding the inverse DFT, $y_1(n)$. The first $L-1$ values of the circular convolution are aliased, and the last



$N - L + 1$ values correspond to the *linear* convolution of $x(n)$ with $h(n)$. Due to the zero padding at the start of $x_1(n)$, these last $N - L + 1$ values are the first $N - L + 1$ values of $y(n)$:

$$y(n) = y_1(n + L - 1) \qquad 0 \leq n \leq N - L$$

3. Let $x_2(n)$ be the N-point sequence that is extracted from $x(n)$ with the first $L - 1$ values overlapping with those of $x_1(n)$.
4. Perform an N-point circular convolution of $x_2(n)$ with $h(n)$ by forming the product $H(k)X_2(k)$ and taking the inverse DFT. The first $L - 1$ values of $y_2(n)$ are discarded and the final $N - L + 1$ values are saved and concatenated with the saved values of $y_1(n)$:

$$y(n + N - L + 1) = y_2(n + L - 1) \qquad 0 \leq n \leq N - L$$

5. Steps 3 and 4 are repeated until all of the values in the linear convolution have been evaluated.

The reason for the name *overlap-save* is that $x(n)$ is partitioned into overlapping sequences of length N and, after performing the N-point circular convolution, only the last $N - L + 1$ values are *saved*.

The Fast Fourier Transform:

The discrete Fourier transform (DFT) is used to perform convolutions. we look at the computational requirements of the DFT and derive some fast algorithms for computing the DFT. These algorithms are known, generically, as fast Fourier fransforms (FFTs). We begin with the radix-2 decimation-in-time FFT. We then look at mixed-radix FFT algorithms and the prime factor FFT.

RADIX-2 FFT ALGORITHMS

The N-point DFT of an N-point sequence $x(n)$ is

$$X(k) = \sum_{n=0}^{N-1} x(n)W_N^{nk}$$

Because $x(n)$ may be either real or complex, evaluating $X(k)$ requires on the order of N complex multiplications and N complex additions for each value of k. Therefore, because there are N values of $X(k)$, computing an N-point DFT requires N^2 complex multiplications and additions.

The basic strategy that is used in the FFT algorithm is one of "divide and conquer," which involves decomposing an N-point DFT into successively smaller DFTs. To see how this works, suppose that the length of $x(n)$ is even (i.e., N is divisible by 2). If $x(n)$ is *decimated* into two sequences of length $N/2$, computing the $N/2$-point DFT of each of these sequences requires approximately $(N/2)^2$ multiplications and the same number of additions. Thus, the two DFTs require $2(N/2)^2 = \frac{1}{2}N^2$ multiplies and adds. Therefore, if it is possible to find the N-point DFT of $x(n)$ from these two $N/2$-point DFTs in fewer than $N^2/2$ operations, a savings has been realized.

Decimation-in-Time FFT

The decimation-in-time FFT algorithm is based on splitting (decimating) $x(n)$ into smaller sequences and finding $X(k)$ from the DFTs of these decimated sequences. This section describes how this decimation leads to an efficient algorithm when the sequence length is a power of 2.

Let $x(n)$ be a sequence of length $N = 2^v$, and suppose that $x(n)$ is split (decimated) into two subsequences, each of length $N/2$. As illustrated in Fig. the first sequence, $g(n)$, is formed from the even-index terms,

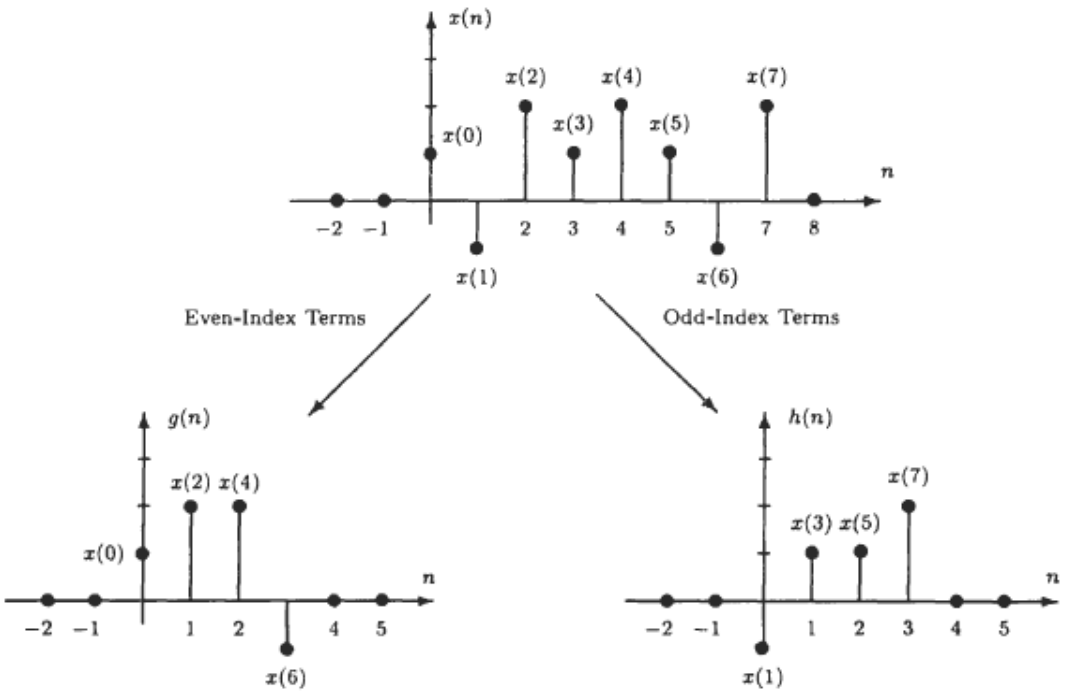
$$g(n) = x(2n) \qquad n = 0, 1, \ldots, \frac{N}{2} - 1$$

and the second, $h(n)$, is formed from the odd-index terms,

$$h(n) = x(2n + 1) \qquad n = 0, 1, \ldots, \frac{N}{2} - 1$$

In terms of these sequences, the N-point DFT of $x(n)$ is

$$\begin{aligned} X(k) &= \sum_{n=0}^{N-1} x(n) W_N^{nk} = \sum_{n \text{ even}} x(n) W_N^{nk} + \sum_{n \text{ odd}} x(n) W_N^{nk} \\ &= \sum_{l=0}^{\frac{N}{2}-1} g(l) W_N^{2lk} + \sum_{l=0}^{\frac{N}{2}-1} h(l) W_N^{(2l+1)k} \end{aligned}$$



Because $W_N^{2lk} = W_{N/2}^{lk}$,

$$X(k) = \sum_{l=0}^{\frac{N}{2}-1} g(l)W_{N/2}^{lk} + W_N^k \sum_{l=0}^{\frac{N}{2}-1} h(l)W_{N/2}^{lk}$$

Note that the first term is the $N/2$-point DFT of $g(n)$, and the second is the $N/2$-point DFT of $h(n)$:

$$X(k) = G(k) + W_N^k H(k) \qquad k = 0, 1, \ldots, N-1$$

Although the $N/2$-point DFTs of $g(n)$ and $h(n)$ are sequences of length $N/2$, the periodicity of the complex exponentials allows us to write

$$G(k) = G\left(k + \frac{N}{2}\right) \qquad H(k) = H\left(k + \frac{N}{2}\right)$$

Therefore, $X(k)$ may be computed from the $N/2$-point DFTs $G(k)$ and $H(k)$. Note that because

$$W_N^{k+N/2} = W_N^k W_N^{N/2} = -W_N^k$$

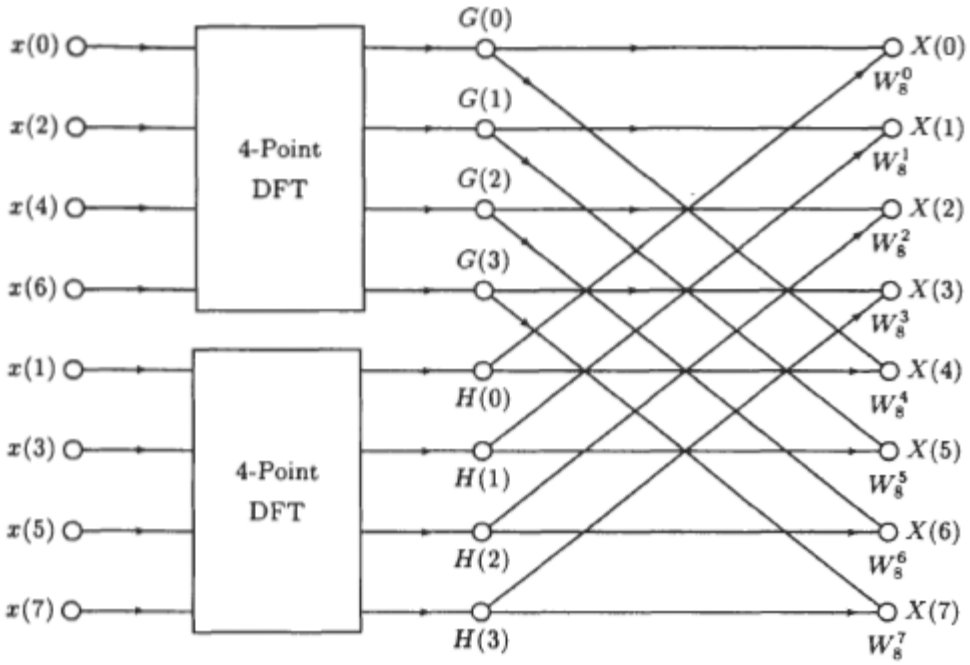
then

$$W_N^{k+\frac{N}{2}} H\left(k + \tfrac{N}{2}\right) = -W_N^k H(k)$$

and it is only necessarv to form the products $W_N^k H(k)$ for $k = 0, 1, \ldots, N/2 - 1$. The complex exponentials multiplying $H(k)$ are called *twiddle factors*. A block diagram showing the computations that are necessary for the first *stage* of an eight-point decimation-in-time FFT is shown in Fig.

If $N/2$ is even, $g(n)$ and $h(n)$ may again be decimated. For example, $G(k)$ may be evaluated as follows:

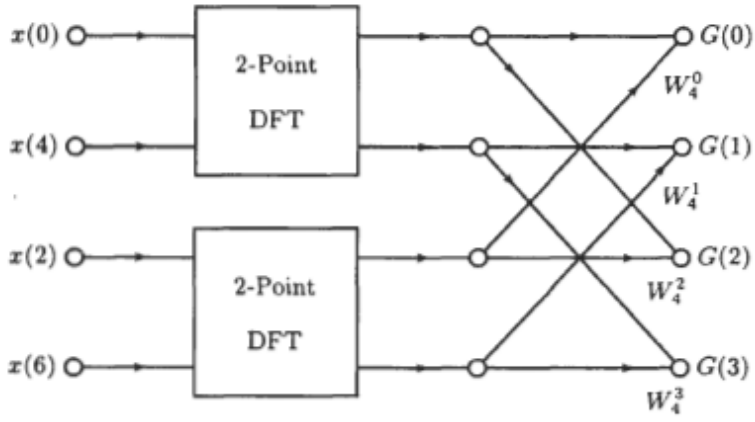
$$G(k) = \sum_{n=0}^{\frac{N}{2}-1} g(n)W_{N/2}^{nk} = \sum_{n \text{ even}}^{\frac{N}{2}-1} g(n)W_{N/2}^{nk} + \sum_{n \text{ odd}}^{\frac{N}{2}-1} g(n)W_{N/2}^{nk}$$

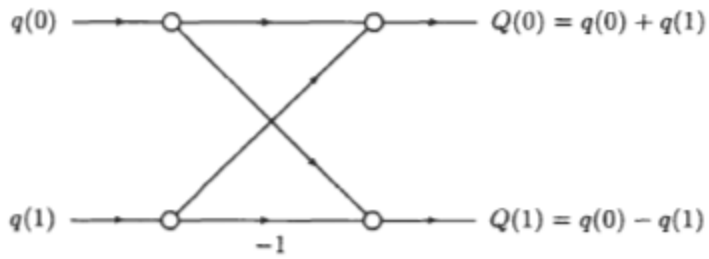


As before, this leads to

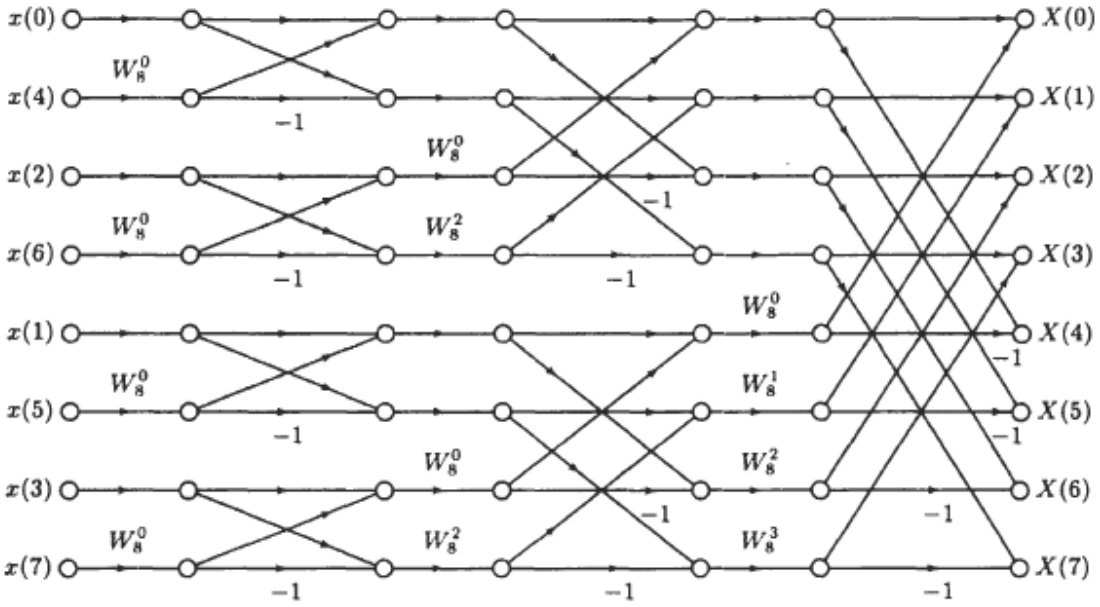
$$G(k) = \sum_{n=0}^{\frac{N}{4}-1} g(2n)W_{N/4}^{nk} + W_{N/2}^{k} \sum_{n=0}^{\frac{N}{4}-1} g(2n+1)W_{N/4}^{nk}$$

where the first term is the $N/4$-point DFT of the even samples of $g(n)$, and the second is the $N/4$-point DFT of the odd samples. A block diagram illustrating this decomposition is shown in Fig. . If N is a power of 2, the decimation may be continued until there are only two-point DFTs of the form shown in Fig.





A two-point DFT.



A complete eight-point radix-2 decimation-in-time FFT.

Computing an N-point DFT using a radix-2 decimation-in-time FFT is much more efficient than calculating the DFT directly. For example, if $N = 2^v$, there are $\log_2 N = v$ *stages* of computation. Because each stage requires $N/2$ complex multiplies by the twiddle factors W_N^r and N complex additions, there are a total of $\frac{1}{2}N \log_2 N$ complex multiplications[1] and $N \log_2 N$ complex additions.

From the structure of the decimation-in-time FFT algorithm, note that once a butterfly operation has been performed on a pair of complex numbers, there is no need to save the input pair. Therefore, the output pair may be stored in the same registers as the input. Thus, only one array of size N is required, and it is said that the computations may be performed *in place*. To perform the computations in place, however, the input sequence $x(n)$ must be stored (or accessed) in nonsequential order as seen in Fig. The *shuffling* of the input sequence that takes place is due to the successive decimations of $x(n)$. The ordering that results corresponds to a bit-reversed indexing of the original sequence. In other words, if the index n is written in binary form, the order in which in the input sequence must be accessed is found by reading the binary representation for n in reverse order as illustrated in the table below for $N = 8$:

n	Binary	Bit-Reversed Binary	n'
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Alternate forms of FFT algorithms may be derived from the decimation-in-time FFT by manipulating the flowgraph and rearranging the order in which the results of each stage of the computation are stored. For example, the nodes of the flowgraph may be rearranged so that the input sequence $x(n)$ is in normal order. What is lost with this reordering, however, is the ability to perform the computations in place.

Decimation-in-Frequency FFT

Another class of FFT algorithms may be derived by decimating the output sequence $X(k)$ into smaller and smaller subsequences. These algorithms are called *decimation-in-frequency* FFTs and may be derived as follows. Let N be a power of 2, $N = 2^v$, and consider separately evaluating the even-index and odd-index samples of $X(k)$. The even samples are

$$X(2k) = \sum_{n=0}^{N-1} x(n) W_N^{2nk}$$

Separating this sum into the first $N/2$ points and the last $N/2$ points, and using the fact that $W_N^{2nk} = W_{N/2}^{nk}$, this becomes

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} x(n) W_{N/2}^{nk} + \sum_{n=N/2}^{N-1} x(n) W_{N/2}^{nk}$$

With a change in the indexing on the second sum we have

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} x(n) W_{N/2}^{nk} + \sum_{n=0}^{\frac{N}{2}-1} x\left(n + \frac{N}{2}\right) W_{N/2}^{(n+\frac{N}{2})k}$$

Finally, because $W_{N/2}^{(n+\frac{N}{2})k} = W_{N/2}^{nk}$,

$$X(2k) = \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) + x\left(n + \frac{N}{2}\right) \right] W_{N/2}^{nk}$$

which is the $N/2$-point DFT of the sequence that is formed by adding the first $N/2$ points of $x(n)$ to the last $N/2$.

Proceeding in the same way for the odd samples of $X(k)$ leads to

$$X(2k+1) = \sum_{n=0}^{\frac{N}{2}-1} W_N^n \left[x(n) - x\left(n + \frac{N}{2}\right) \right] W_{N/2}^{nk} \tag{7.4}$$

A flowgraph illustrating this first stage of decimation is shown in Fig. 7-7. As with the decimation-in-time FFT, the decimation may be continued until only two-point DFTs remain. A complete eight-point decimation-in-frequency FFT is shown in Fig. 7-8. The complexity of the decimation-in-frequency FFT is the same as the decimation-in-time, and the computations may be performed in place. Finally, note that although the input sequence $x(n)$ is in normal order, the frequency samples $X(k)$ are in bit-reversed order.

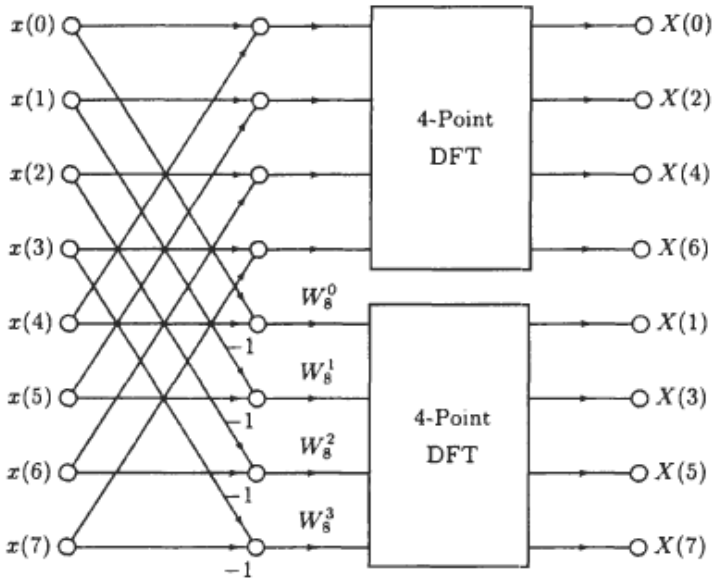


Fig. 7-7. An eight-point decimation-in-frequency FFT algorithm after the first stage of decimation.

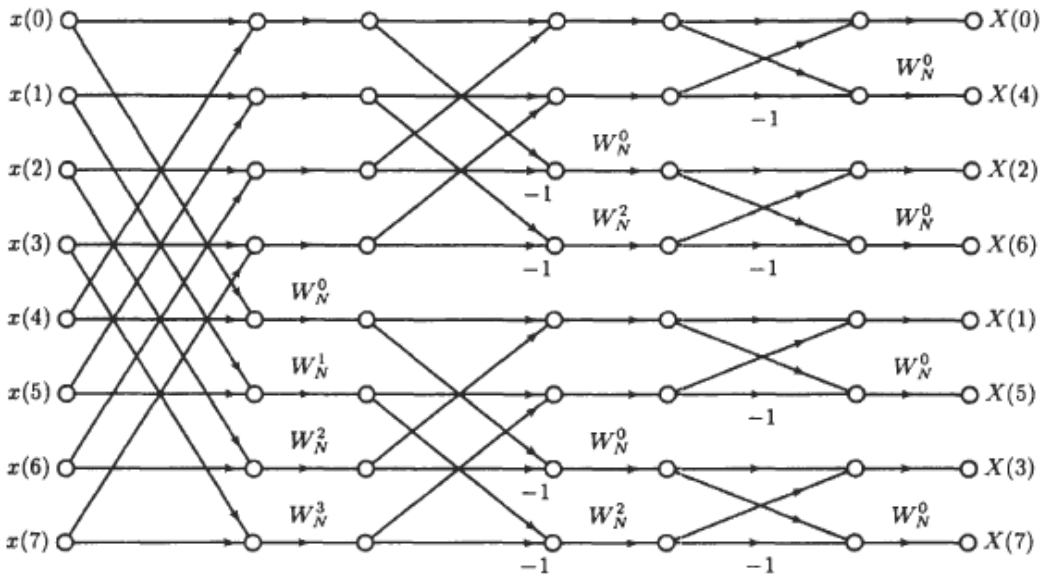


Fig. 7-8. Eight-point radix-2 decimation-in-frequency FFT.

STRUCTURES FOR IIR SYSTEMS

The input $x(n)$ and output $y(n)$ of a causal IIR filter with a rational system function

$$H(z) = \frac{B(z)}{A(z)} = \frac{\sum_{k=0}^{q} b(k)z^{-k}}{1 + \sum_{k=1}^{p} a(k)z^{-k}}$$

is described by the linear constant coefficient difference equation

$$y(n) = \sum_{k=0}^{q} b(k)x(n-k) - \sum_{k=1}^{p} a(k)y(n-k)$$

In the following sections, several different implementations of this system are presented, including the direct form structures, the cascade and parallel forms, and the transposed filter structures.

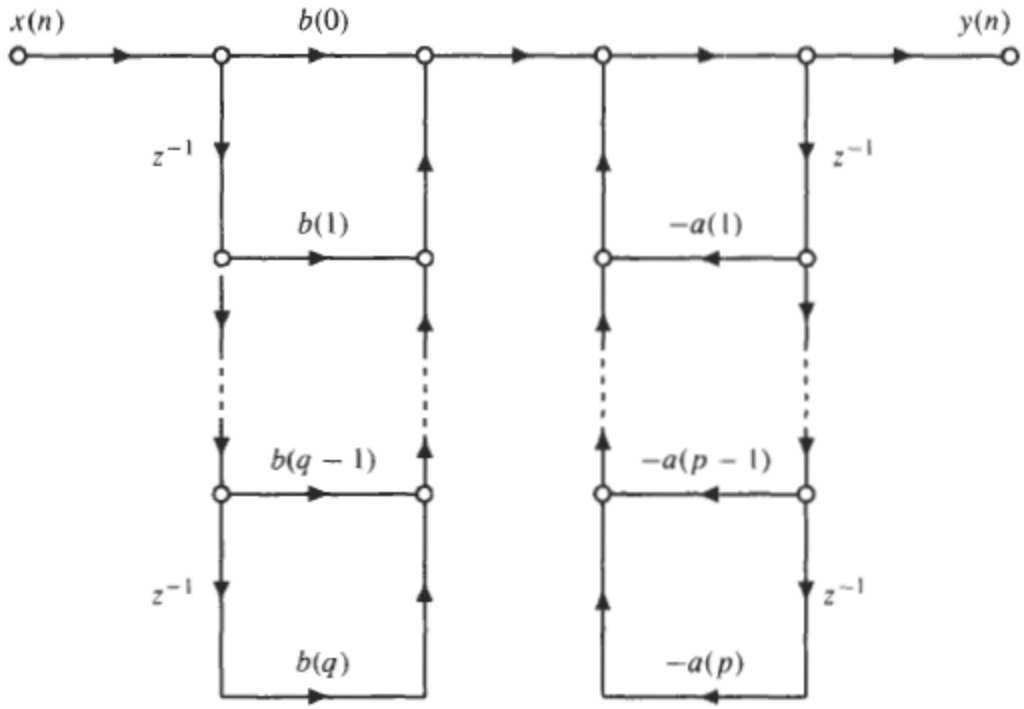
8.4.1 Direct Form

There are two direct form filter structures, referred to as *direct form I* and *direct form II*. The direct form I structure is an implementation that results when Eq. (*8.3*) is written as a pair of difference equations as follows:

$$w(n) = \sum_{k=0}^{q} b(k)x(n-k)$$

$$y(n) = w(n) - \sum_{k=1}^{p} a(k)y(n-k)$$

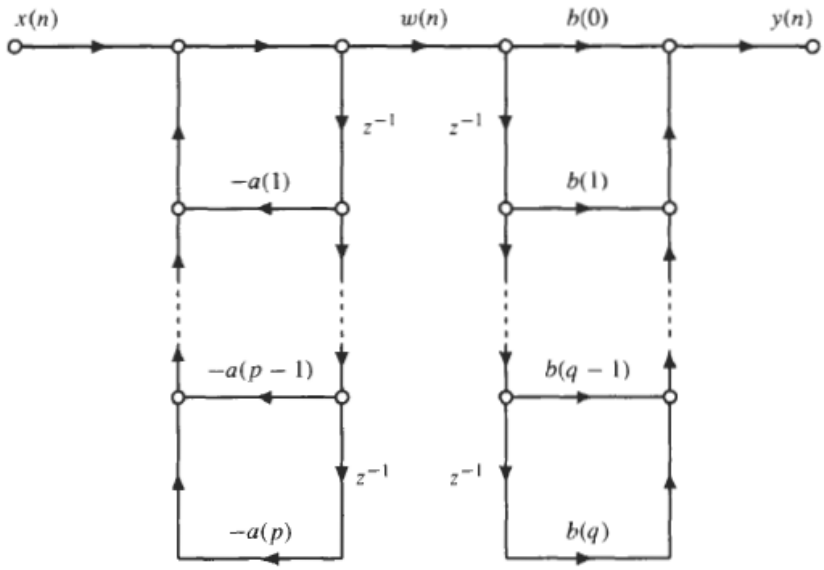
The first equation corresponds to an FIR filter with input $x(n)$ and output $w(n)$, and the second equation corresponds to an all-pole filter with input $w(n)$ and output $y(n)$. Therefore, this pair of equations represents a



Direct form I realization of an IIR filter.

The *direct form II* structure is obtained by reversing the order of the cascade of $B(z)$ and $1/A(z)$ as illustrated in Fig. . With this implementation, $x(n)$ is first filtered with the all-pole filter $1/A(z)$ and then with $B(z)$:

$$Y(z) = B(z)\left[\frac{1}{A(z)}X(z)\right]$$



Reversing the order of the cascade in the direct form I filter structure.

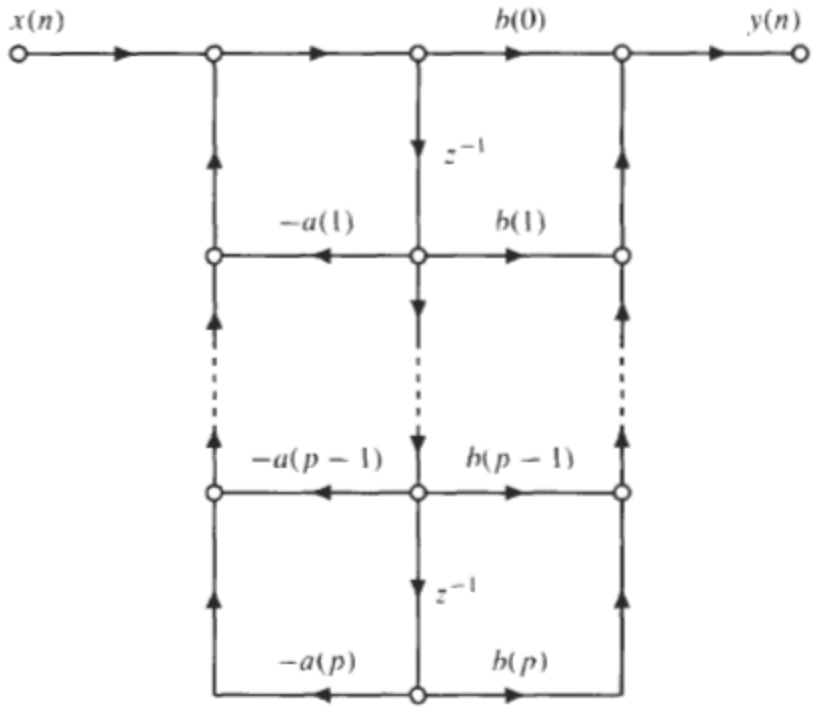
This structure may be simplified by noting that the two sets of delays are delaying the same sequence. Therefore, they may be combined as illustrated in Fig. for the case in which $p = q$. The computational requirements for a direct form II structure are as follows:

Number of multiplications: $p + q + 1$ per output sample

Number of additions: $p + q$ per output sample

Number of delays: $\max(p, q)$

The direct form II structure is said to be *canonic* because it uses the minimum number of delays for a given $H(z)$.



Direct form II realization of an IIR filter with $p = q$.

8.4.2 Cascade Structure

The cascade structure is derived by factoring the numerator and denominator polynomials of $H(z)$:

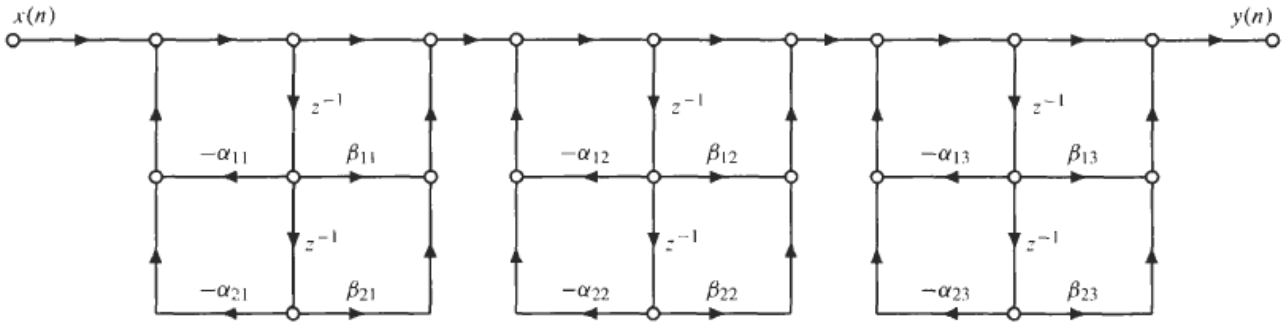
$$H(z) = \frac{\sum_{k=0}^{q} b(k)z^{-k}}{1 + \sum_{k=1}^{p} a(k)z^{-k}} = A \prod_{k=1}^{\max\{p,q\}} \frac{1 - \beta_k z^{-1}}{1 - \alpha_k z^{-1}}$$

This factorization corresponds to a *cascade* of first-order filters, each having one pole and one zero. In general, the coefficients α_k and β_k will be complex. However, if $h(n)$ is real, the roots of $H(z)$ will occur in complex

conjugate pairs, and these complex conjugate factors may be combined to form second-order factors with *real* coefficients:

$$H_k(z) = \frac{1 + \beta_{1k} z^{-1} + \beta_{2k} z^{-2}}{1 + \alpha_{1k} z^{-1} + \alpha_{2k} z^{-2}}$$

A sixth-order IIR filter implemented as a cascade of three second-order systems in direct form II is shown in Fig.



A sixth-order IIR filter implemented as a cascade of three direct form II second-order systems.

There is considerable flexibility in how a system may be implemented in cascade form. For example, there are different *pairings* of the poles and zeros and different ways in which the sections may be *ordered.*

Parallel Structure

An alternative to factoring $H(z)$ is to expand the system function using a partial fraction expansion. For example, with

$$H(z) = \frac{\sum_{k=0}^{q} b(k) z^{-k}}{1 + \sum_{k=1}^{p} a(k) z^{-k}} = A \frac{\prod_{k=1}^{q} (1 - \beta_k z^{-1})}{\prod_{k=1}^{p} (1 - \alpha_k z^{-1})}$$

if $p > q$ and $\alpha_i \neq \alpha_k$ (the roots of the denominator polynomial are distinct), $H(z)$ may be expanded as a sum of p first-order factors as follows:

$$H(z) = \sum_{k=1}^{p} \frac{A_k}{1 - \alpha_k z^{-1}}$$

where the coefficients A_k and α_k are, in general, complex. This expansion corresponds to a sum of p first-order system functions and may be realized by connecting these systems in parallel. If $h(n)$ is real, the poles of $H(z)$ will occur in complex conjugate pairs, and these complex roots in the partial fraction expansion may be combined to form second-order systems with real coefficients:

$$H(z) = \sum_{k=1}^{N_s} \frac{\gamma_{0k} + \gamma_{1k} z^{-1}}{1 + \alpha_{1k} z^{-1} + \alpha_{2k} z^{-2}}$$

Shown in Fig. is a sixth-order filter implemented as a parallel connection of three second-order direct form II systems. If $p \leq q$, the partial fraction expansion will also contain a term of the form

$$c_0 + c_1 z^{-1} + \cdots + c_{q-p} z^{-(q-p)}$$

which is an FIR filter that is placed in parallel with the other terms in the expansion of $H(z)$.

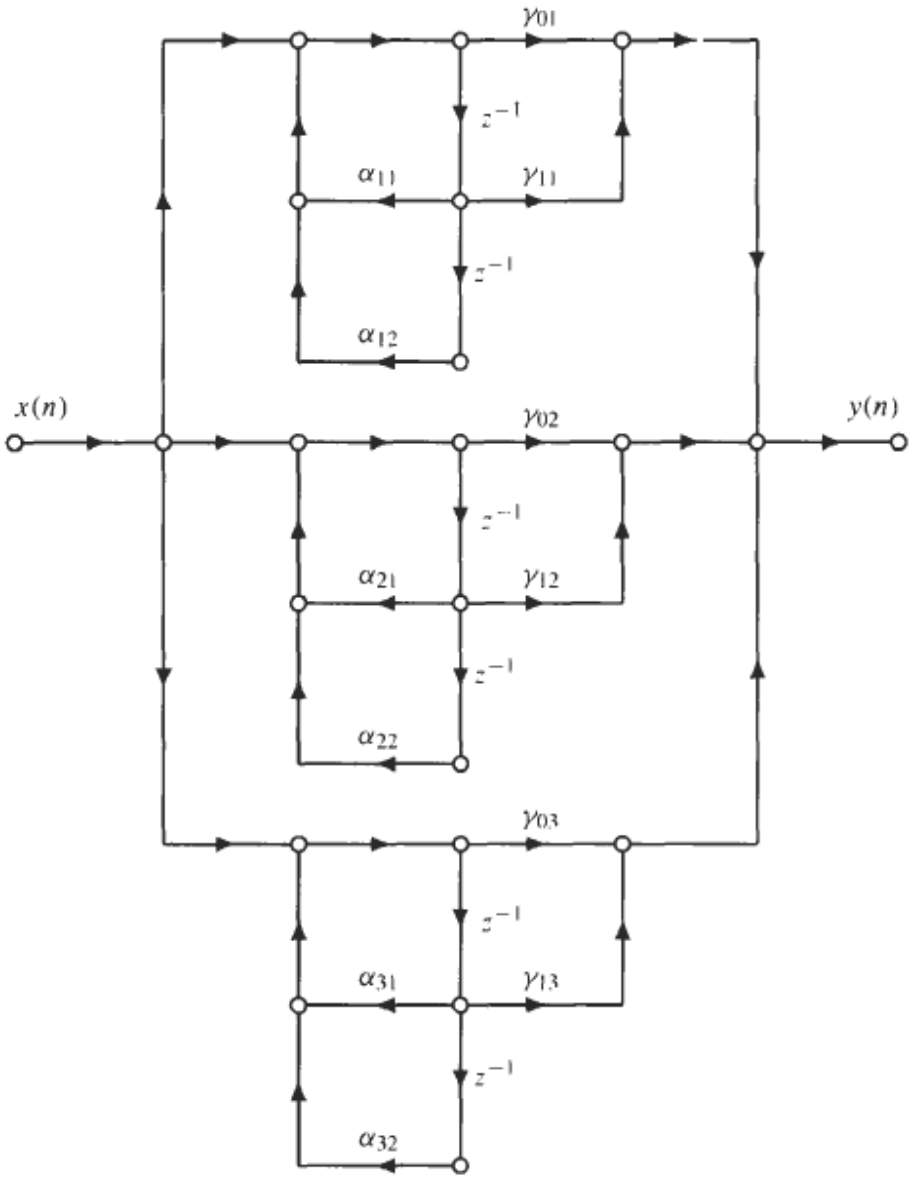


Fig. 8-10. A sixth-order IIR filter implemented as a parallel connection of three second-order direct form II structures.

Example1:

Implement the system

$$H(z) = \frac{4 + \frac{9}{4}z^{-1} - \frac{1}{4}z^{-2}}{1 + \frac{1}{4}z^{-1} - \frac{1}{8}z^{-2}}$$

as a parallel network of first-order direct form structures.

Factoring the denominator of the system function, we find

$$H(z) = \frac{4 + \frac{9}{4}z^{-1} - \frac{1}{4}z^{-2}}{\left(1 - \frac{1}{4}z^{-1}\right)\left(1 + \frac{1}{2}z^{-1}\right)}$$

To implement $H(z)$ as a parallel network of first-order filters, we must express $H(z)$ as a sum of first-order factors using a partial fraction expansion. Because the order of the numerator is equal to the order of the denominator, this expansion will contain a constant term,

$$H(z) = C + \frac{A}{1 - \frac{1}{4}z^{-1}} + \frac{B}{1 + \frac{1}{2}z^{-1}}$$

To find the value of C, we divide the numerator polynomial by the denominator as follows:

$$\begin{array}{r} 2 \\ -\frac{1}{8}z^{-2} + \frac{1}{4}z^{-1} + 1 \;\overline{\left| \begin{array}{r} -\frac{1}{4}z^{-2} + \frac{9}{4}z^{-1} + 4 \\ -\frac{1}{4}z^{-2} + \frac{1}{2}z^{-1} + 2 \\ \hline \frac{7}{4}z^{-1} + 2 \end{array} \right.} \end{array}$$

Therefore, $C = 2$, and we may write $H(z)$ as follows:

$$H(z) = 2 + \frac{\frac{7}{4}z^{-1} + 2}{\left(1 - \frac{1}{4}z^{-1}\right)\left(1 + \frac{1}{2}z^{-1}\right)}$$

Finally, with

$$G(z) = \frac{\frac{7}{4}z^{-1} + 2}{\left(1 - \frac{1}{4}z^{-1}\right)\left(1 + \frac{1}{2}z^{-1}\right)}$$

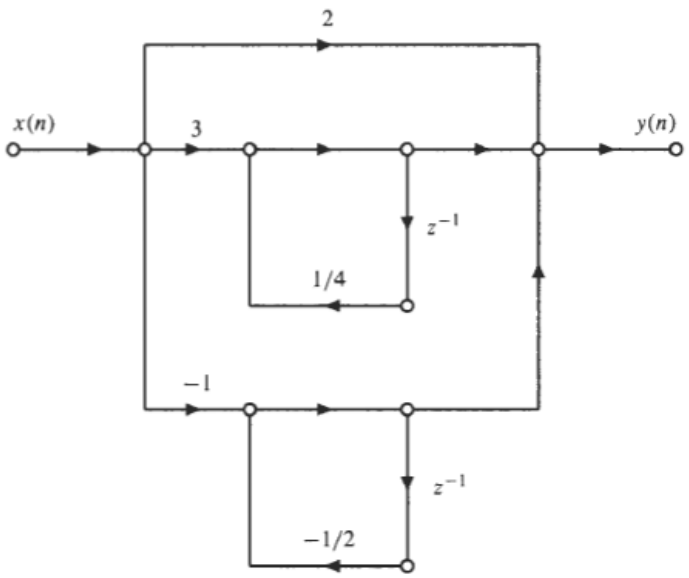
we have, for the coefficients A and B,

$$A = \left[\left(1 - \frac{1}{4}z^{-1}\right)G(z)\right]_{z^{-1}=4} = \left.\frac{\frac{7}{4}z^{-1} + 2}{1 + \frac{1}{2}z^{-1}}\right|_{z^{-1}=4} = 3$$

and $$B = \left[\left(1 + \tfrac{1}{2}z^{-1}\right)G(z)\right]_{z^{-1}=-2} = \left.\frac{\frac{7}{4}z^{-1} + 2}{1 - \frac{1}{4}z^{-1}}\right|_{z^{-1}=-2} = -1$$

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

and the parallel network for this system is as shown below.



Example 2:

8.10 Consider the causal linear shift-invariant filter with system function

$$H(z) = \frac{1 + 0.875z^{-1}}{(1 + 0.2z^{-1} + 0.9z^{-2})(1 - 0.7z^{-1})}$$

Draw a signal flowgraph for this system using

(*a*) Direct form I

(*b*) Direct form II

(*c*) A cascade of first- and second-order systems realized in direct form II

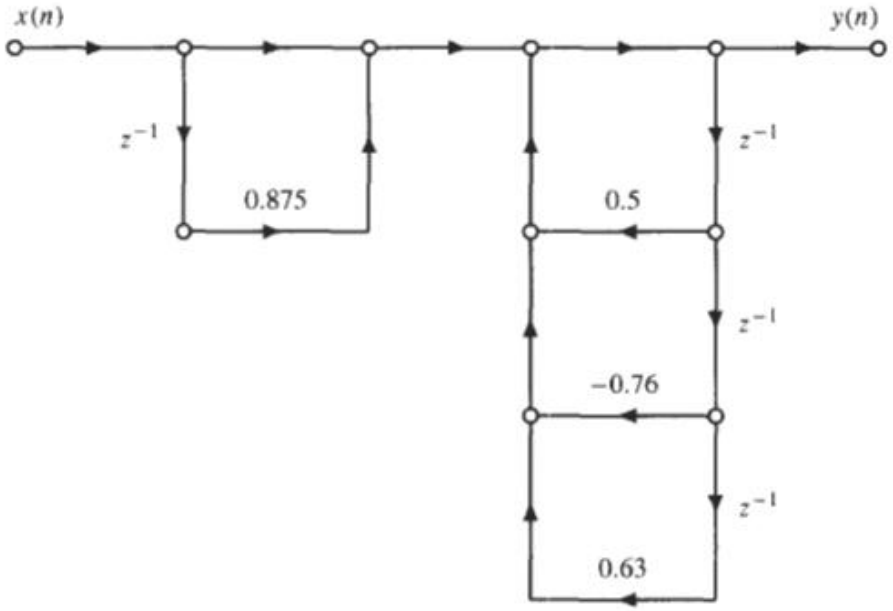
(*d*) A cascade of first- and second-order systems realized in transposed direct form II

(*e*) A parallel connection of first- and second-order systems realized in direct form II

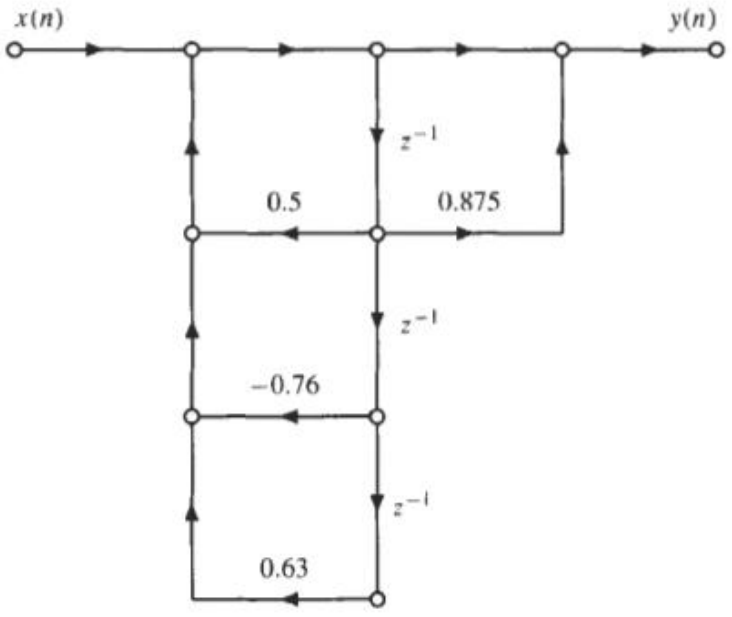
(*a*) Writing the system function as a ratio of polynomials in z^{-1},

$$H(z) = \frac{1 + 0.875z^{-1}}{1 - 0.5z^{-1} + 0.76z^{-2} - 0.63z^{-3}}$$

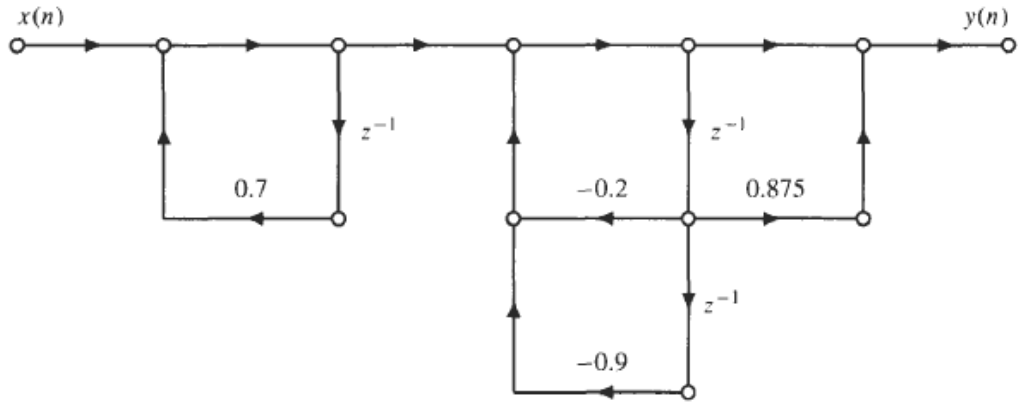
it follows that the direct form I realization of $H(z)$ is as follows:



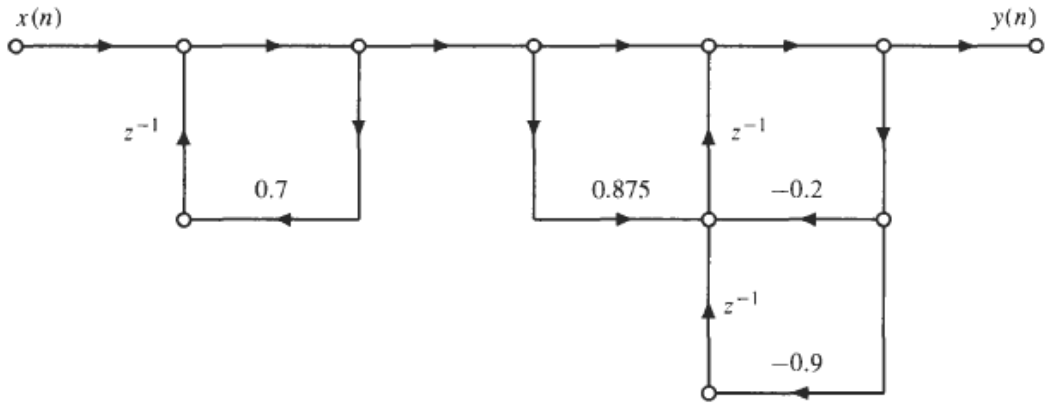
(*b*) For a direct form II realization of $H(z)$, we have



(*c*) Using a cascade of first- and second-order systems realized in direct form II, we have a choice of either pairing the zero with the first-order factor in the denominator or with the second-order factor. Although it does not make a difference from a computational point of view, because the zero is closer to the pair of complex poles than to the pole at $z = 0.7$, we will pair the zero with the second-order factor. With this pairing, the realization of $H(z)$ is as follows:



(*d*) If we change the direct form II systems in part (*c*) to transposed direct form II, we have the realization shown below.



(*e*) For a parallel structure, $H(z)$ must be expanded using a partial fraction expansion:

$$H(z) = \frac{1 + 0.875z^{-1}}{(1 + 0.2z^{-1} + 0.9z^{-2})(1 - 0.7z^{-1})} = \frac{A + Bz^{-1}}{1 + 0.2z^{-1} + 0.9z^{-2}} + \frac{C}{1 - 0.7z^{-1}}$$

The constants A, B, and C may be found as follows. Recombining the two terms in the partial fraction expansion as follows,

$$\begin{aligned} H(z) &= \frac{A + Bz^{-1}}{1 + 0.2z^{-1} + 0.9z^{-2}} + \frac{C}{1 - 0.7z^{-1}} \\ &= \frac{(A + C) + (B + 0.2C - 0.7A)z^{-1} + (0.9C - 0.7B)z^{-2}}{(1 + 0.2z^{-1} + 0.9z^{-2})(1 - 0.7z^{-1})} \end{aligned}$$

and equating the coefficients in the numerator of this expression with the numerator of $H(z)$, we have the following three equations in the three unknowns A, B, and C:

$$A + C = 1$$
$$B + 0.2C - 0.7A = 0.875$$
$$0.9C - 0.7B = 0$$

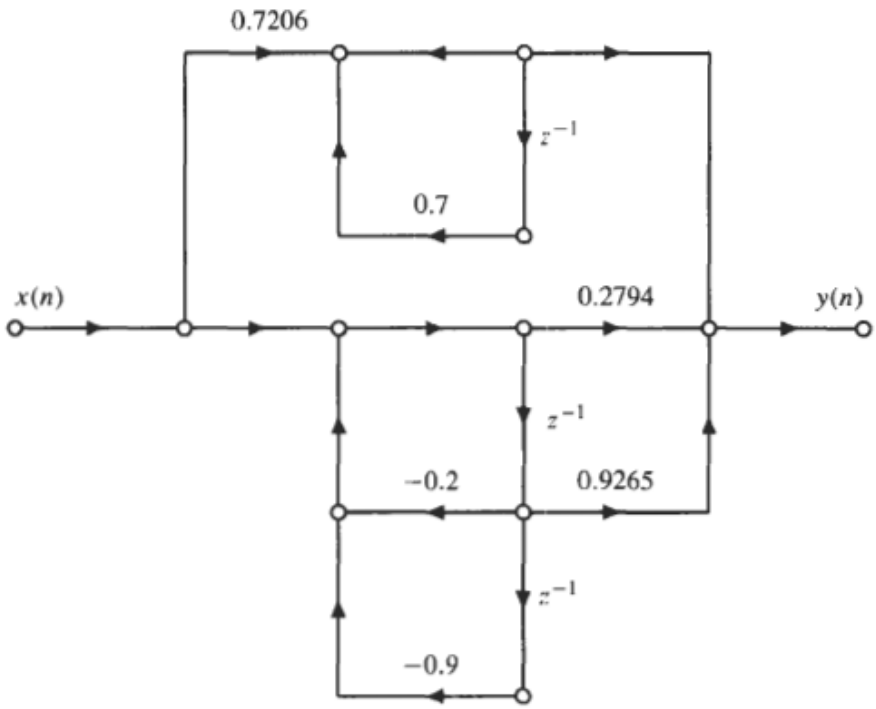
Solving for A, B, and C we find

$$A = 0.2794 \qquad B = 0.9265 \qquad C = 0.7206$$

and, therefore, the partial fraction expansion is

$$H(z) = \frac{0.2794 + 0.9265z^{-1}}{1 + 0.2z^{-1} + 0.9z^{-2}} + \frac{0.7206}{1 - 0.7z^{-1}}$$

Thus, a parallel structure for $H(z)$ is shown below.



IIR FILTER DESIGN

There are two general approaches used to design IIR digital filters. The most common is to design an analog IIR filter and then map it into an equivalent digital filter because the art of analog filter design is highly advanced. Therefore, it is prudent to consider optimal ways for mapping these filters into the discrete-time domain. Furthermore, because there are powerful design procedures that facilitate the design of analog filters, this approach

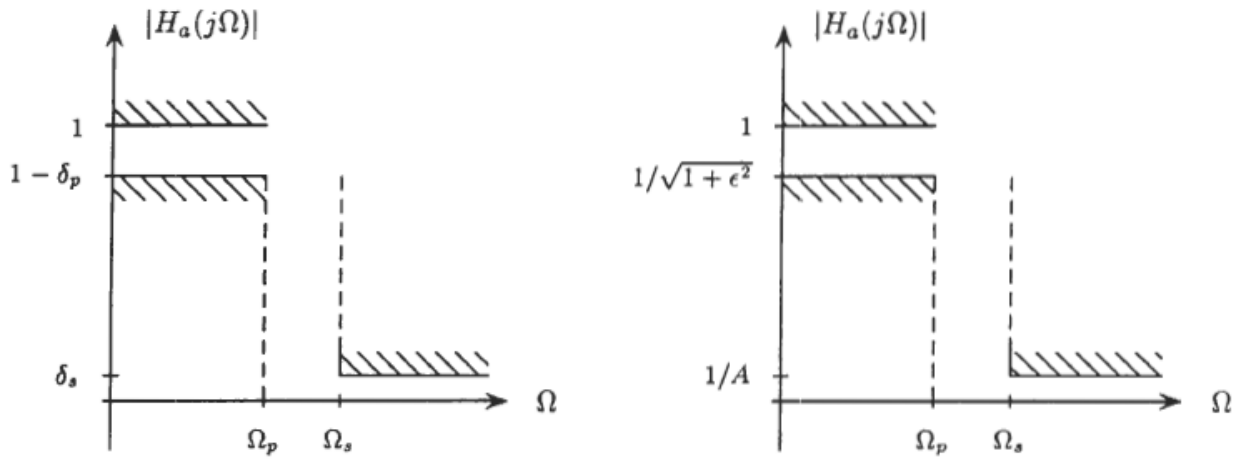
to IIR filter design is relatively simple. The second approach to design IIR digital filters is to use an algorithmic design procedure, which generally requires the use of a computer to solve a set of linear or nonlinear equations. These methods may be used to design digital filters with arbitrary frequency response characteristics for which no analog filter prototype exists or to design filters when other types of constraints are imposed on the design.

In this section, we consider the approach of mapping analog filters into digital filters. Initially, the focus will be on the design of digital low-pass filters from analog low-pass filters. Techniques for transforming these designs into more general frequency selective filters will then be discussed.

Analog Low-Pass Filter Prototypes

To design an IIR digital low-pass filter from an analog low-pass filter, we must first know how to design an analog low-pass filter. Historically, most analog filter approximation methods were developed for the design of passive systems having a gain less than or equal to 1. Therefore, a typical set of specifications for these filters is as shown in Fig. with the passband specifications having the form

$$1 - \delta_p \leq |H_a(j\Omega)| \leq 1$$



(*a*) Specifications in terms of δ_p and δ_s.

(*b*) Specifications in terms of ϵ and A.

Two different conventions for specifying the passband and stopband deviations for an analog low-pass filter.

Another convention that is commonly used is to describe the passband and stopband constraints in terms of the parameters ϵ and A as illustrated in Fig. 9-5(b). Two auxiliary parameters of interest are the *discrimination factor*,

$$d = \left[\frac{(1-\delta_p)^{-2}-1}{\delta_s^{-2}-1}\right]^{1/2} = \frac{\epsilon}{\sqrt{A^2-1}}$$

and the *selectivity factor*

$$k = \frac{\Omega_p}{\Omega_s}$$

The three most commonly used analog low-pass filters are the Butterworth, Chebyshev, and elliptic filters. These filters are described below.

Butterworth Filter

A low-pass Butterworth filter is an all-pole filter with a squared magnitude response given by

$$|H_a(j\Omega)|^2 = \frac{1}{1+(j\Omega/j\Omega_c)^{2N}}$$

The parameter N is the order of the filter (number of poles in the system function), and Ω_c is the 3-dB cutoff frequency. The magnitude of the frequency response may also be written as

$$|H_a(j\Omega)|^2 = \frac{1}{1+\epsilon^2(j\Omega/j\Omega_p)^{2N}}$$

where

$$\epsilon = \left(\frac{\Omega_p}{\Omega_c}\right)^N$$

Steps to design Butterworth IIR Filter:

1. Find the values for the selectivity factor, k, and the discrimination factor, d, from the filter specifications.
2. Determine the order of the filter required to meet the specifications using the design formula

$$N \geq \frac{\log d}{\log k}$$

3. Set the 3-dB cutoff frequency, Ω_c, to any value in the range

$$\Omega_p[(1-\delta_p)^{-2}-1]^{-1/2N} \le \Omega_c \le \Omega_s\left[\delta_s^{-2}-1\right]^{-1/2N}$$

4. Synthesize the system function of the Butterworth filter from the poles of

$$G_a(s) = H_a(s)H_a(-s) = \frac{1}{1+(s/j\Omega_c)^{2N}}$$

that lie in the left-half s-plane. Thus,

$$H_a(s) = \prod_{k=0}^{N-1} \frac{-s_k}{s-s_k}$$

where $$s_k = \Omega_c \exp\left\{j\frac{(N+1+2k)\pi}{2N}\right\} \qquad k = 0,\ 1,\ \ldots,\ N-1$$

Table The Coefficients in the System Function of a Normalized Butterworth Filter ($\Omega_c = 1$) for Orders $1 \le N \le 8$

N	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
1	1.0000							
2	1.4142	1.0000						
3	2.0000	2.0000	1.0000					
4	2.6131	3.4142	2.6131	1.0000				
5	3.2361	5.2361	5.2361	3.2361	1.0000			
6	3.8637	7.4641	9.1416	7.4641	3.8637	1.0000		
7	4.4940	10.0978	14.5918	14.5918	10.0978	4.4940	1.0000	
8	5.1258	13.1371	21.8462	25.6884	21.8462	13.1372	5.1258	1.0000

Example1:

Let us design a low-pass Butterworth filter to meet the following specifications:

$$f_p = 6\ \text{kHz} \qquad f_s = 10\ \text{kHz} \qquad \delta_p = \delta_s = 0.1$$

First, we compute the discrimination and selectivity factors:

$$d = \left[\frac{(1-\delta_p)^{-2}-1}{\delta_s^{-2}-1}\right]^{1/2} = 0.0487 \qquad k = \frac{\Omega_p}{\Omega_s} = \frac{f_p}{f_s} = 0.6$$

Because

$$N \geq \frac{\log d}{\log k} = 5.92$$

it follows that the minimum filter order is $N = 6$. With

$$f_p[(1-\delta_p)^{-2}-1]^{-1/2N} = 6770$$

and

$$f_s\left[\delta_s^{-2}-1\right]^{-1/2N} = 6819$$

the center frequency, f_c, may be any value in the range

$$6770 \leq f_c \leq 6819$$

The system function of the Butterworth filter may then be found using Eq. by first constructing a sixth-order normalized Butterworth filter from Table

$$H_a(s) = \frac{1}{s^6 + 3.8637s^5 + 7.4641s^4 + 9.1416s^3 + 7.4641s^2 + 3.8637s + 1}$$

and then replacing s with s/Ω_c so that the cutoff frequency is Ω_c instead of unity

Chebyshev Filters

Chebyshev filters are defined in terms of the Chebyshev polynomials:

$$T_N(x) = \begin{cases} \cos(N \cos^{-1} x) & |x| \leq 1 \\ \cosh(N \cosh^{-1} x) & |x| > 1 \end{cases}$$

These polynomials may be generated recursively as follows,

$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x) \qquad k \geq 1$$

with $T_0(x) = 1$ and $T_1(x) = x$. The following properties of the Chebyshev polynomials follow from Eq.

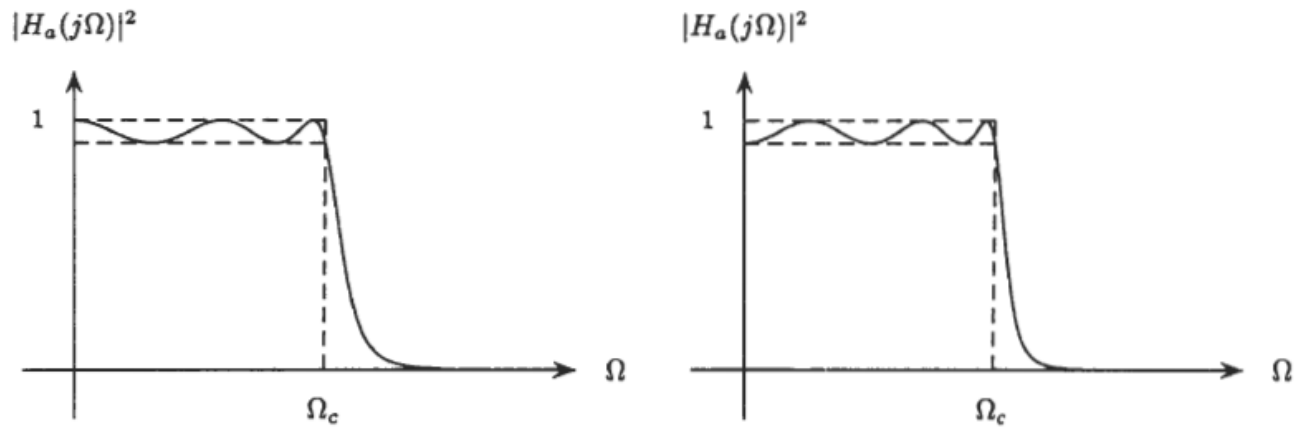
1. For $|x| \leq 1$ the polynomials are bounded by 1 in magnitude, $|T_N(x)| \leq 1$, and oscillate between ± 1. For $|x| > 1$, the polynomials increase monotonically with x.
2. $T_N(1) = 1$ for all N.
3. $T_N(0) = \pm 1$ for N even, and $T_N(0) = 0$ for N odd.
4. All of the roots of $T_N(x)$ are in the interval $-1 \leq x \leq 1$.

Type I filter:

There are two types of Chebyshev filters. A type I Chebyshev filter is all-pole with an equiripple passband and a monotonically decreasing stopband. The magnitude of the frequency response is

$$|H_a(j\Omega)|^2 = \frac{1}{1 + \epsilon^2 T_N^2(\Omega/\Omega_p)}$$

where N is the order of the filter, Ω_p is the passband cutoff frequency, and ϵ is a parameter that controls the passband ripple amplitude. Because $T_N^2(\Omega/\Omega_p)$ varies between 0 and 1 for $|\Omega| < \Omega_p$, $|H_a(j\Omega)|^2$ oscillates between 1 and $1/(1+\epsilon^2)$. As the order of the filter increases, the number of oscillations (ripples) in the passband increases, and the transition width between the passband and stopband becomes narrower. Examples are given in Fig. for $N = 5, 6$.



(*a*) Odd order ($N = 5$). (*b*) Even order ($N = 6$).

Frequency response of Chebyshev type I filter for orders $N = 5$ and $N = 6$.

The system function of a type I Chebyshev filter has the form

$$H_a(s) = H_a(0) \prod_{k=0}^{N-1} \frac{-s_k}{s - s_k}$$

where $H_a(0) = (1 - \epsilon^2)^{-1/2}$ if N is even, and $H_a(0) = 1$ if N is odd.

Steps to design type I chebyshev filter:

1. Find the values for the selectivity factor, k, and the discrimination factor, d.
2. Determine the filter order using the formula

$$N \geq \frac{\cosh^{-1}(1/d)}{\cosh^{-1}(1/k)}$$

3. Form the rational function

$$G_a(s) = H_a(s)H_a(-s) = \frac{1}{1 + \epsilon^2 T_N^2(s/j\Omega_p)}$$

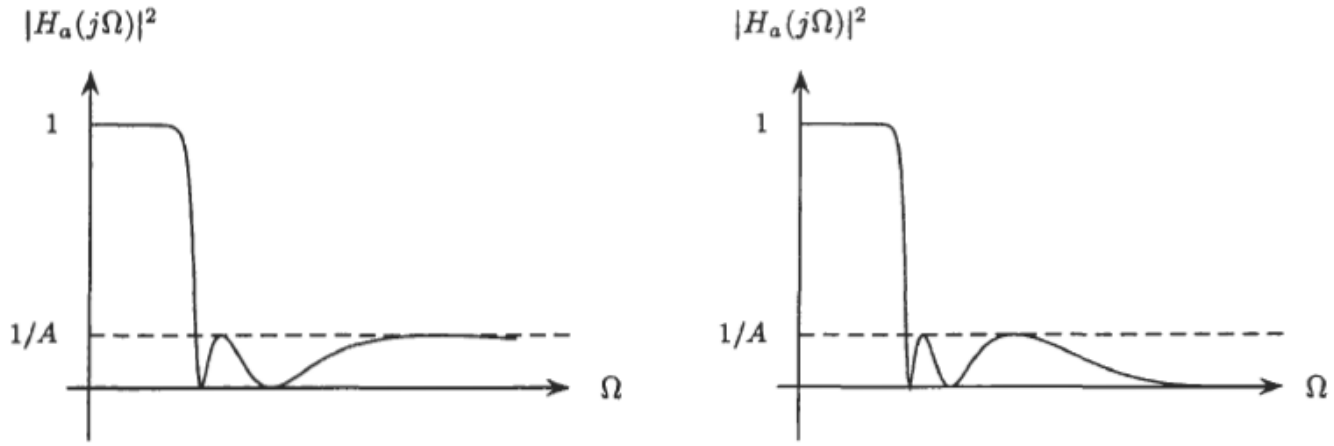
where $\epsilon = [(1 - \delta_p)^{-2} - 1]^{1/2}$, and construct the system function $H_a(s)$ by taking the N poles of $G_a(s)$ that lie in the left-half s-plane.

Type II filter:

A type II Chebyshev filter, unlike a type I filter, has a monotonic passband and an equiripple stopband, and the system function has both poles and zeros. The magnitude of the frequency response is

$$|H_a(j\Omega)|^2 = \frac{1}{1 + \epsilon^2[T_N(\Omega_s/\Omega_p)/T_N(\Omega_s/\Omega)]^2}$$

where N is the order of the filter, Ω_p is the passband cutoff frequency, Ω_s is the stopband cutoff frequency, and ϵ is the parameter that controls the stopband ripple amplitude. Again, as the order N is increased, the number of ripples increases and the transition width becomes narrower. Examples are given in Fig. for $N = 5, 6$.



(*a*) Odd order ($N = 5$). (*b*) Even order ($N = 6$).

Frequency response of a Chebyshev type II filter for orders $N = 5$ and $N = 6$.

The system function of a type II Chebyshev filter has the form

$$H_a(s) = \prod_{k=0}^{N-1} \frac{a_k}{b_k} \frac{s - b_k}{s - a_k}$$

The poles are located at

$$a_k = \frac{\Omega_s^2}{s_k}$$

where s_k for $k = 0, 1, \ldots, N-1$ are the poles of a type I Chebyshev filter. The zeros b_k lie on the $j\Omega$-axis at the frequencies for which $T_N(\Omega_s/\Omega) = 0$.

The procedure for designing a type II Chebyshev filter is the same as for a type I filter, except that

$$\epsilon = \left(\delta_s^{-2} - 1\right)^{-1/2}$$

Example 1:

If $H_a(s)$ is a third-order type I Chebyshev low-pass filter with a cutoff frequency $\Omega_p = 1$ and $\epsilon = 0.1$, find $H_a(s)H_a(-s)$.

The magnitude of the frequency response squared for an Nth-order type I Chebyshev filter is

$$|H_a(j\Omega)|^2 = \frac{1}{1 + \epsilon^2 T_N^2(\Omega/\Omega_p)}$$

where $T_N(x)$ is an Nth-order Chebyshev polynomial that is defined recursively as follows,

$$T_{k+1}(x) = 2xT_k(x) - T_{k-1}(x) \qquad k \geq 1$$

with $T_0(x) = 1$ and $T_1(x) = x$. Therefore, to find the third-order Chebyshev polynomial, we first find $T_2(x)$ as follows,

$$T_2(x) = 2xT_1(x) - T_0(x) = 2x^2 - 1$$

and then we have

$$T_3(x) = 2xT_2(x) - T_1(x) = 4x^3 - 2x - x = x(4x^2 - 3)$$

Thus, the denominator polynomial in $|H_a(j\Omega)|^2$ is

$$\begin{aligned} 1 + \epsilon^2 T_3^2\left(\frac{\Omega}{\Omega_p}\right) &= 1 + 0.01[\Omega(4\Omega^2 - 3)]^2 \\ &= 1 + 0.01\Omega^2(16\Omega^4 - 24\Omega^2 + 9) \end{aligned}$$

and we have

$$|H_a(j\Omega)|^2 = \frac{1}{1 + 0.09\Omega^2 - 0.24\Omega^4 + 0.16\Omega^6}$$

Because

$$|H_a(j\Omega)|^2 = [H_a(s)H_a(-s)]_{s=j\Omega}$$

to find the rational function

$$G_a(s) = H_a(s)H_a(-s)$$

we make the substitution $\Omega = s/j$ in $|H_a(j\Omega)|^2$ as follows:

$$G_a(s) = \frac{1}{1 + 0.09(s/j)^2 - 0.24(s/j)^4 + 0.16(s/j)^6} = \frac{1}{1 - 0.09s^2 - 0.24s^4 - 0.16s^6}$$

Design of IIR Filters from Analog Filters

The design of a digital filter from an analog prototype requires that we transform $h_a(t)$ to $h(n)$ or $H_a(s)$ to $H(z)$. A mapping from the s-plane to the z-plane may be written as

$$H(z) = H_a(s)\big|_{s=m(z)}$$

where $s = m(z)$ is the mapping function. In order for this transformation to produce an acceptable digital filter, the mapping $m(z)$ should have the following properties:

1. The mapping from the $j\Omega$-axis to the unit circle, $|z| = 1$, should be one to one and *onto* the unit circle in order to preserve the frequency response characteristics of the analog filter.
2. Points in the left-half s-plane should map to points *inside* the unit circle to preserve the stability of the analog filter.
3. The mapping $m(z)$ should be a rational function of z so that a rational $H_a(s)$ is mapped to a rational $H(z)$.

Methods to map analog filters into digital filters:
1.Impulse invariance.
2.Bilinear transformation.

Impulse Invariance

With the *impulse invariance* method, a digital filter is designed by sampling the impulse response of an analog filter:

$$h(n) = h_a(nT_s)$$

From the sampling theorem, it follows that the frequency response of the digital filter, $H(e^{j\omega})$, is related to the frequency response $H_a(j\Omega)$ of the analog filter as follows:

$$H(e^{j\omega}) = \frac{1}{T_s}\sum_{k=-\infty}^{\infty} H_a\left(j\frac{\omega}{T_s} + j\frac{2\pi k}{T_s}\right)$$

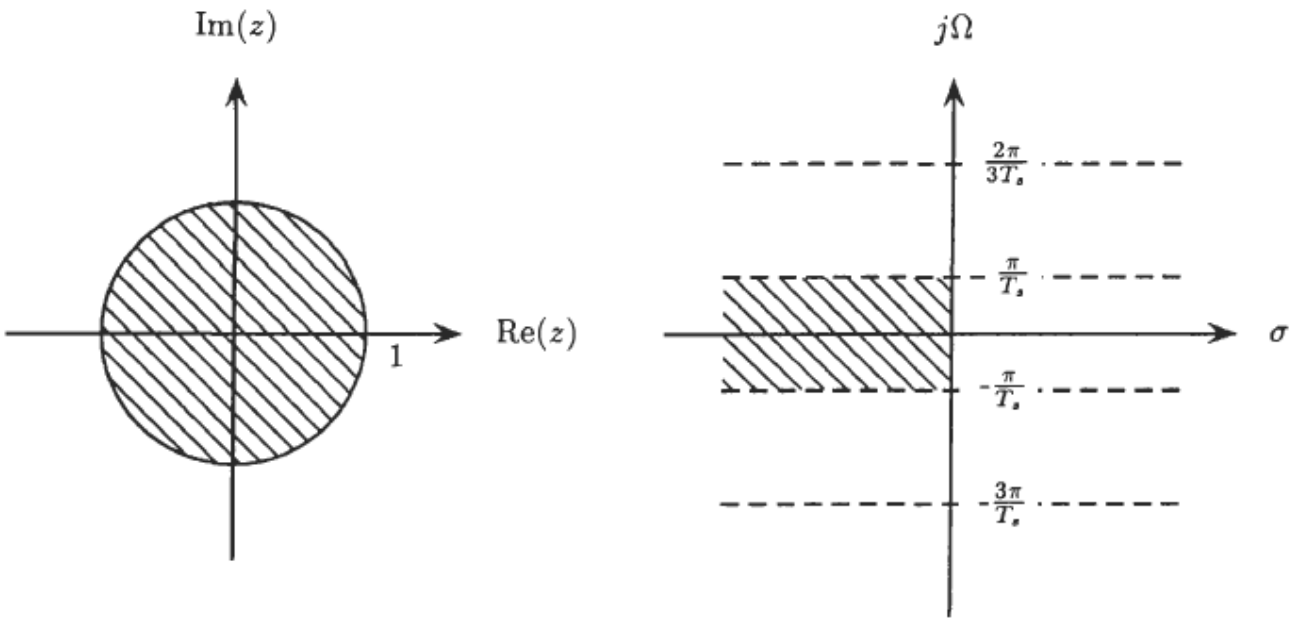
More generally, this may be extended into the complex plane as follows:

$$H(z)\big|_{z=e^{sT_s}} = \frac{1}{T_s}\sum_{k=-\infty}^{\infty} H_a\left(s + j\frac{2\pi k}{T_s}\right)$$

The mapping between the s-plane and the z-plane is illustrated in Fig. Note that although the $j\Omega$-axis maps *onto* the unit circle, the mapping is not one to one. In particular, each interval of length $2\pi/T_s$ along the $j\Omega$-axis is mapped onto the unit circle (i.e., the frequency response is aliased). In addition, each point in the left-half s-plane is mapped to a point *inside* the unit circle. Specifically, strips of width $2\pi/T_s$ map *onto* the z-plane. If the frequency response of the analog filter, $H_a(j\Omega)$, is sufficiently bandlimited, then

$$H(e^{j\omega}) \approx \frac{1}{T_s}H_a\left(\frac{j\omega}{T_s}\right)$$

Although the impulse invariance may produce a reasonable design in some cases, this technique is essentially limited to bandlimited analog filters.



Properties of the s-plane to z-plane mapping in the impulse invariance method.

To see how poles and zeros of an analog filter are mapped using the impulse invariance method, consider an analog filter that has a system function

$$H_a(s) = \sum_{k=1}^{p} \frac{A_k}{s - s_k}$$

The impulse response, $h_a(t)$, is

$$h_a(t) = \sum_{k=1}^{p} A_k e^{s_k t} u(t)$$

Therefore, the digital filter that is formed using the impulse invariance technique is

$$h(n) = h_a(nT_s) = \sum_{k=1}^{p} A_k e^{s_k nT_s} u(nT_s) = \sum_{k=1}^{p} A_k (e^{s_k T_s})^n u(n)$$

and the system function is

$$H(z) = \sum_{k=1}^{p} \frac{A_k}{1 - e^{s_k T_s} z^{-1}}$$

Thus, a pole at $s = s_k$ in the analog filter is mapped to a pole at $z = e^{s_k T_s}$ in the digital filter,

$$\frac{1}{s - s_k} \Longrightarrow \frac{1}{1 - e^{s_k T_s} z^{-1}}$$

The zeros, however, do not get mapped in any obvious way.

The Bilinear Transformation

The bilinear transformation is a mapping from the s-plane to the z-plane defined by

$$s = \frac{2}{T_s}\frac{1 - z^{-1}}{1 + z^{-1}}$$

Given an analog filter with a system function $H_a(s)$, the digital filter is designed as follows:

$$H(z) = H_a\left(\frac{2}{T_s}\frac{1 - z^{-1}}{1 + z^{-1}}\right)$$

The bilinear transformation is a rational function that maps the left-half s-plane *inside* the unit circle and maps the $j\Omega$-axis in a one-to-one manner *onto* the unit circle. However, the relationship between the $j\Omega$-axis and the unit circle is highly nonlinear and is given by the *frequency warping function*

$$\omega = 2\arctan\left(\frac{\Omega T_s}{2}\right)$$

As a result of this warping, the bilinear transformation will only preserve the magnitude response of analog filters that have an ideal response that is piecewise constant. Therefore, the bilinear transformation is generally only used in the design of frequency selective filters.

The parameter T_s in the bilinear transformation is normally included for historical reasons. However, it does not enter into the design process, because it only scales the $j\Omega$-axis in the frequency warping function, and this scaling may be done in the specification of the analog filter. Therefore, T_s may be set to any value to simplify the design procedure.

The steps involved in the design of a digital low-pass filter with a passband cutoff frequency ω_p, stopband cutoff frequency ω_s, passband ripple δ_p, and stopband ripple δ_s are as follows:

1. *Prewarp* the passband and stopband cutoff frequencies of the digital filter, ω_p and ω_s, using the inverse of Eq. (*9.12*) to determine the passband and cutoff frequencies of the analog low-pass filter. With $T_s = 2$, the prewarping function is
$$\Omega = \tan\left(\frac{\omega}{2}\right)$$
2. Design an analog low-pass filter with the cutoff frequencies found in step 1 and a passband and stopband ripple δ_p and δ_s, respectively.
3. Apply the bilinear transformation to the filter designed in step 2.

Example1:

Let us design a first-order digital low-pass filter with a 3-dB cutoff frequency of $\omega_c = 0.25\pi$ by applying the bilinear transformation to the analog Butterworth filter

$$H_a(s) = \frac{1}{1 + s/\Omega_c}$$

Because the 3-dB cutoff frequency of the Butterworth filter is Ω_c, for a cutoff frequency $\omega_c = 0.25\pi$ in the digital filter, we must have

$$\Omega_c = \frac{2}{T_s}\tan\left(\frac{0.25\pi}{2}\right) = \frac{0.828}{T_s}$$

Therefore, the system function of the analog filter is

$$H_a(s) = \frac{1}{1 + sT_s/0.828}$$

Applying the bilinear transformation to the analog filter gives

$$H(z) = H_a(s)\Big|_{s=\frac{2}{T_s}\frac{1-z^{-1}}{1+z^{-1}}} = \frac{1}{1 + (2/0.828)[(1 - z^{-1})/(1 + z^{-1})]} = 0.2920\frac{1 + z^{-1}}{1 - 0.4159z^{-1}}$$

Note that the parameter T_s does not enter into the design.

Frequency Transformations

The preceding section considered the design of digital low-pass filters from analog low-pass filters. There are two approaches that may be used to design other types of frequency selective filters, such as high-pass, bandpass, or bandstop filters. The first is to design an analog low-pass filter and then apply a frequency transformation to map the analog filter into the desired frequency selective prototype. This analog prototype is then mapped to a digital filter using a suitable s-plane to z-plane mapping. Table 1 provides a list of some analog-to-analog transformations.

Table 1 The Transformation of an Analog Low-pass Filter with a 3-dB Cutoff Frequency Ω_p to Other Frequency Selective Filters

Transformation	Mapping	New Cutoff Frequencies
Low-pass	$s \rightarrow \frac{\Omega_p}{\Omega'_p} s$	Ω'_p
High-pass	$s \rightarrow \frac{\Omega_p \Omega'_p}{s}$	Ω'_p
Bandpass	$s \rightarrow \Omega_p \frac{s^2 + \Omega_l \Omega_u}{s(\Omega_u - \Omega_l)}$	Ω_l, Ω_u
Bandstop	$s \rightarrow \Omega_p \frac{s(\Omega_u - \Omega_l)}{s^2 + \Omega_l \Omega_u}$	Ω_l, Ω_u

The second approach that may be used is to design an analog low-pass filter, map it into a digital filter using a suitable s-plane to z-plane mapping, and then apply an appropriate frequency transformation in the discrete-time domain to produce the desired frequency selective digital filter. Table 2 provides a list of some digital-to-digital transformations. The two approaches do not always result in the same design. For example, although the second approach could be used to design a high-pass filter using the impulse invariance technique, with the first approach the design would be unacceptable due to the aliasing that would occur when sampling the analog high-pass filter.

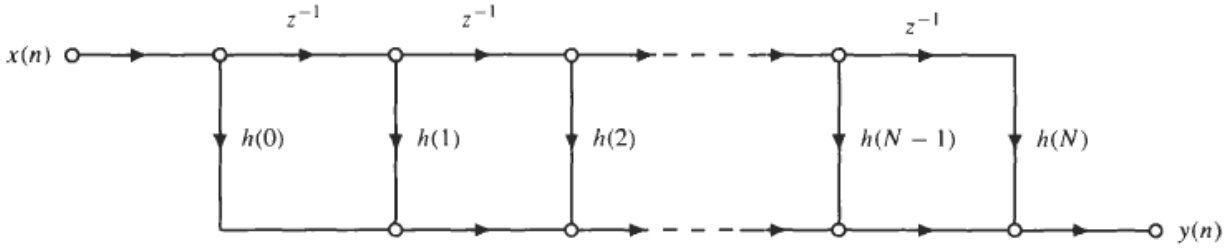
Table 2 The Transformation of a Digital Low-Pass Filter with a Cutoff Frequency ω_c to Other Frequency Selective Filters

Filter Type	Mapping	Design Parameters
Low-pass	$z^{-1} \rightarrow \dfrac{z^{-1} - \alpha}{1 - \alpha z^{-1}}$	$\alpha = \dfrac{\sin[(\omega_c - \omega_c')/2]}{\sin[(\omega_c + \omega_c')/2]}$ ω_c' = desired cutoff frequency
High-pass	$z^{-1} \rightarrow -\dfrac{z^{-1} + \alpha}{1 + \alpha z^{-1}}$	$\alpha = -\dfrac{\cos[(\omega_c + \omega_c')/2]}{\cos[(\omega_c - \omega_c')/2]}$ ω_c' = desired cutoff frequency
Bandpass	$z^{-1} \rightarrow -\dfrac{z^{-2} - [2\alpha\beta/(\beta+1)]z^{-1} + [(\beta-1)/(\beta+1)]}{[(\beta-1)/(\beta+1)]z^{-2} - [2\alpha\beta/(\beta+1)]z^{-1} + 1}$	$\alpha = \dfrac{\cos[(\omega_{c2} + \omega_{c1})/2]}{\cos[(\omega_{c2} - \omega_{c1})/2]}$ $\beta = \cot[(\omega_{c2} - \omega_{c1})/2]\tan(\omega_c/2)$ ω_{c1} = desired lower cutoff frequency ω_{c2} = desired upper cutoff frequency
Bandstop	$z^{-1} \rightarrow \dfrac{z^{-2} - [2\alpha/(\beta+1)]z^{-1} + [(1-\beta)/(1+\beta)]}{[(1-\beta)/(1+\beta)]z^{-2} - [2\alpha/(\beta+1)]z^{-1} + 1}$	$\alpha = \dfrac{\cos[(\omega_{c1} + \omega_{c2})/2]}{\cos[(\omega_{c1} - \omega_{c2})/2]}$ $\beta = \tan[(\omega_{c2} - \omega_{c1})/2]\tan(\omega_c/2)$ ω_{c1} = desired lower cutoff frequency ω_{c2} = desired upper cutoff frequency

UNIT – IV

1 Direct Form

The most common way to implement an FIR filter is in *direct form* using a *tapped delay line* as shown in th figure below.



This structure requires $N + 1$ multiplications, N additions, and N delays. However, if there are some symmetrie in the unit sample response, it may be possible to reduce the number of multiplications (see the section on linea phase filters).

2 *Cascade Form*

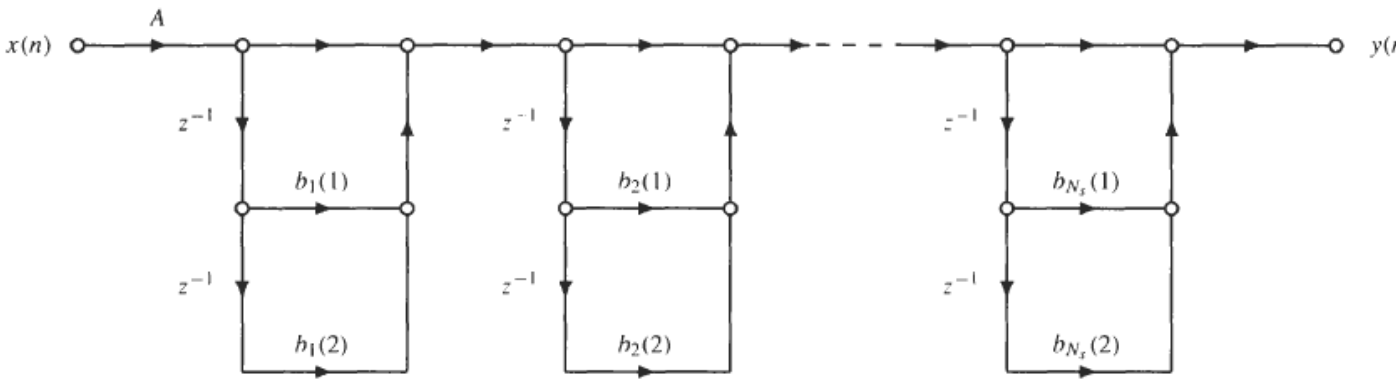
For a causal FIR filter, the system function may be factored into a product of first-order factors,

$$H(z) = \sum_{n=0}^{N} h(n)z^{-n} = A\prod_{k=1}^{N}(1 - \alpha_k z^{-1})$$

where α_k for $k = 1, \ldots, N$ are the zeros of $H(z)$. If $h(n)$ is real, the complex roots of $H(z)$ occur in compl conjugate pairs, and these conjugate pairs may be combined to form second-order factors with real coefficien

$$H(z) = A\prod_{k=1}^{N_s}[1 + b_k(1)z^{-1} + b_k(2)z^{-2}]$$

Written in this form, $H(z)$ may be implemented as a cascade of second-order FIR filters as illustrated in Fig.



An FIR filter implemented as a cascade of second-order systems.

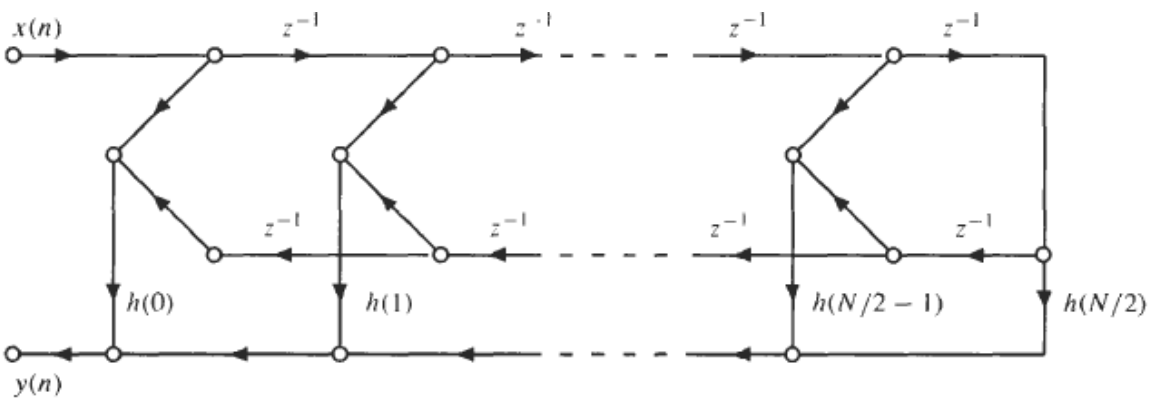
.3 *Linear Phase Filters*

Linear phase filters have a unit sample response that is either symmetric, $h(n) = h(N - n)$
or antisymmetric $h(n) = -h(N - n)$

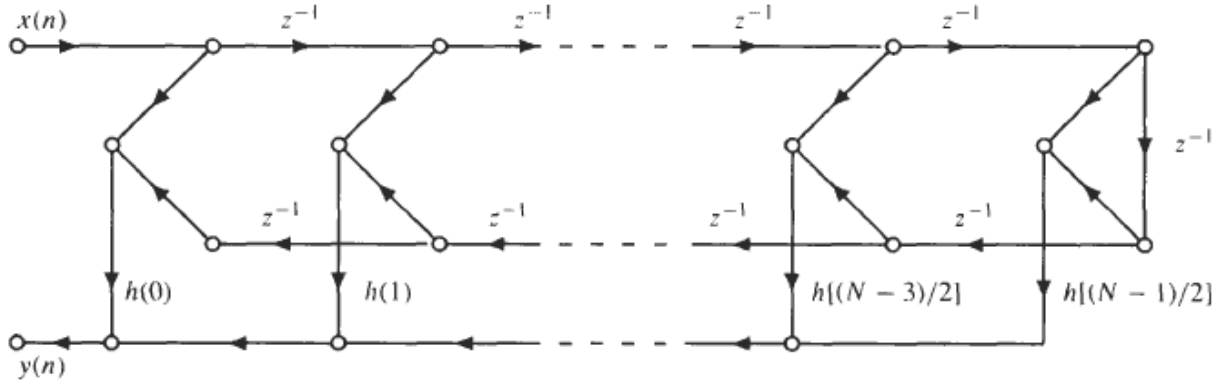
This symmetry may be exploited to simplify the network structure. For example, if N is even and $h(n)$ symmetric (type I filter),

$$y(n) = \sum_{k=0}^{N} h(k)x(n-k) = \sum_{k=0}^{\frac{N}{2}-1} h(k)[x(n-k) + x(n-N+k)] + h\left(\frac{N}{2}\right)x\left(n - \frac{N}{2}\right)$$

Therefore, forming the sums $[x(n-k) + x(n-N+k)]$ prior to multiplying by $h(k)$ reduces the number o multiplications. The resulting structure is shown in Fig. a If N is odd and $h(n)$ is symmetric (type II filter), the structure is as shown in Fig. b There are similar structures for the antisymmetric (types III and IV) linear phase filters.



(a)



(b)

Direct form implementations for linear phase filters. (*a*) Type I. (*b*) Type II.

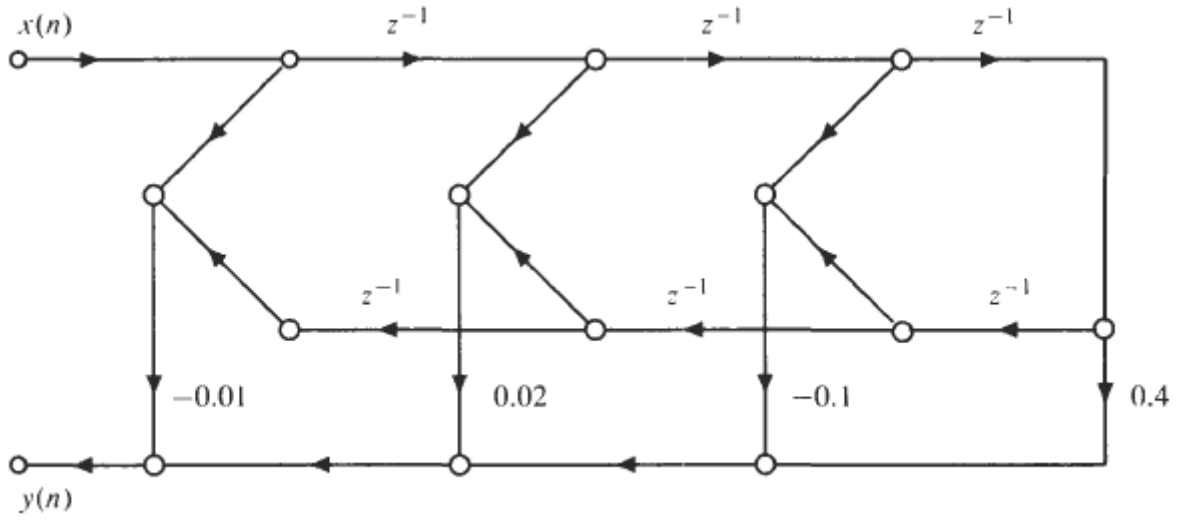
Example:

A linear shift-invariant system has a unit sample response given by

$h(0) = -0.01 \quad h(1) = 0.02 \quad h(2) = -0.10 \quad h(3) = 0.40$

$h(4) = -0.10 \quad h(5) = 0.02 \quad h(6) = -0.01$

Draw a signal flowgraph for this system that requires the minimum number of multiplications.



FILTER SPECIFICATIONS

Before a filter can be designed, a set of filter specifications must be defined. For example, suppose that we would like to design a low-pass filter with a cutoff frequency ω_c. The frequency response of an ideal low-pass filter with linear phase and a cutoff frequency ω_c is

$$H_d(e^{j\omega}) = \begin{cases} e^{-j\alpha\omega} & |\omega| \le \omega_c \\ 0 & \omega_c < |\omega| \le \pi \end{cases}$$

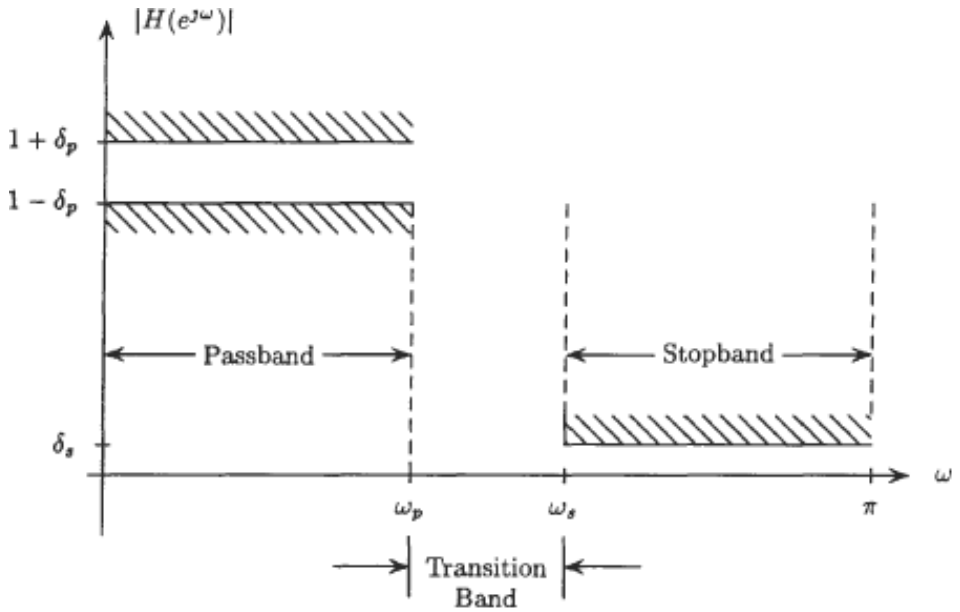
which has a unit sample response

$$h_d(n) = \frac{\sin(n-\alpha)\omega_c}{\pi(n-\alpha)}$$

Because this filter is unrealizable (noncausal and unstable), it is necessary to relax the ideal constraints on the frequency response and allow some deviation from the ideal response. The specifications for a low-pass filter will typically have the form

$$\begin{aligned} 1-\delta_p < |H(e^{j\omega})| \le 1+\delta_p \qquad & 0 \le |\omega| < \omega_p \\ |H(e^{j\omega})| \le \delta_s \qquad & \omega_s \le |\omega| < \pi \end{aligned}$$

as illustrated in Fig. Thus, the specifications include the passband cutoff frequency, ω_p, the stopband cutoff frequency, ω_s, the passband deviation, δ_p, and the stopband deviation, δ_s. The passband and stopband deviations



Filter specifications for a low-pass filter.

are often given in decibels (dB) as follows:

$$\alpha_p = -20\log(1-\delta_p)$$

and

$$\alpha_s = -20\log(\delta_s)$$

The interval $[\omega_p, \omega_s]$ is called the *transition band*.

Once the filter specifications have been defined, the next step is to design a filter that meets these specifications.

FIR FILTER DESIGN

The frequency response of an Nth-order causal FIR filter is

$$H(e^{j\omega}) = \sum_{n=0}^{N} h(n)e^{-jn\omega}$$

and the design of an FIR filter involves finding the coefficients $h(n)$ that result in a frequency response that satisfies a given set of filter specifications. FIR filters have two important advantages over IIR filters. First, they are guaranteed to be stable, even after the filter coefficients have been quantized. Second, they may be easily constrained to have (generalized) linear phase. Because FIR filters are generally designed to have linear phase, in the following we consider the design of linear phase FIR filters.

Linear Phase FIR Design Using Windows

Let $h_d(n)$ be the unit sample response of an ideal frequency selective filter with linear phase,

$$H_d(e^{j\omega}) = A(e^{j\omega})e^{-j(\alpha\omega-\beta)}$$

Because $h_d(n)$ will generally be infinite in length, it is necessary to find an FIR approximation to $H_d(e^{j\omega})$. With the window design method, the filter is designed by windowing the unit sample response,

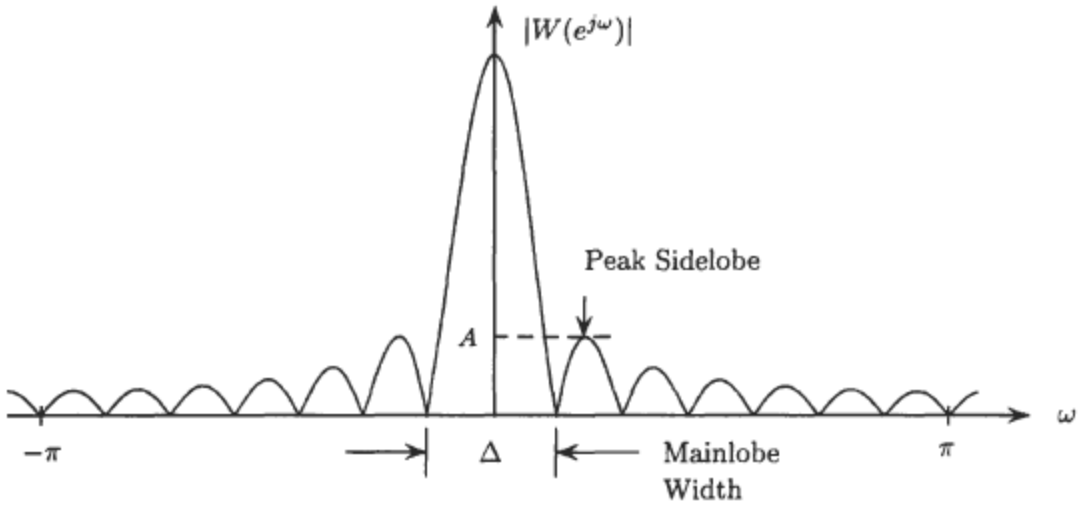
$$h(n) = h_d(n)w(n)$$

where $w(n)$ is a finite-length window that is equal to zero outside the interval $0 \le n \le N$ and is symmetric about its midpoint:

$$w(n) = w(N-n)$$

$H_d(e^{j\omega})$, is determined by two factors

1. The width of the main lobe of $W(e^{j\omega})$.
2. The peak side-lobe amplitude of $W(e^{j\omega})$.



The DTFT of a typical window, which is characterized by the width of its main lobe, Δ, and the peak amplitude of its side lobes, A, relative to the amplitude of $W(e^{j\omega})$ at $\omega = 0$.

Some general properties of windows are as follows:

1. As the length N of the window increases, the width of the main lobe decreases, which results in a decrease in the transition width between passbands and stopbands. This relationship is given approximately by

$$N\Delta f = c$$

where Δf is the transition width, and c is a parameter that depends on the window.

2. The peak side-lobe amplitude of the window is determined by the shape of the window, and it is essentially independent of the window length.
3. If the window shape is changed to decrease the side-lobe amplitude, the width of the main lobe will generally increase.

Some Common Windows

Rectangular	$w(n) = \begin{cases} 1 & 0 \le n \le N \\ 0 & \text{else} \end{cases}$
Hanning[1]	$w(n) = \begin{cases} 0.5 - 0.5\cos\left(\frac{2\pi n}{N}\right) & 0 \le n \le N \\ 0 & \text{else} \end{cases}$
Hamming	$w(n) = \begin{cases} 0.54 - 0.46\cos\left(\frac{2\pi n}{N}\right) & 0 \le n \le N \\ 0 & \text{else} \end{cases}$
Blackman	$w(n) = \begin{cases} 0.42 - 0.5\cos\left(\frac{2\pi n}{N}\right) + 0.08\cos\left(\frac{4\pi n}{N}\right) & 0 \le n \le N \\ 0 & \text{else} \end{cases}$

The Peak Side-Lobe Amplitude of Some Common Windows and the Approximate Transition Width and Stopband Attenuation of an Nth-Order Low-Pass Filter Designed Using the Given Window.

Window	Side-Lobe Amplitude (dB)	Transition Width (Δf)	Stopband Attenuation (dB)
Rectangular	−13	$0.9/N$	−21
Hanning	−31	$3.1/N$	−44
Hamming	−41	$3.3/N$	−53
Blackman	−57	$5.5/N$	−74

Kaiser window:

$$w(n) = \frac{I_0[\beta(1 - [(n-\alpha)/\alpha]^2)^{1/2}]}{I_0(\beta)} \qquad 0 \le n \le N$$

where $\alpha = N/2$, and $I_0(\cdot)$ is a zeroth-order modified Bessel function of the first kind, which may be easily generated using the power series expansion

$$I_0(x) = 1 + \sum_{k=1}^{\infty} \left[\frac{(x/2)^k}{k!}\right]^2$$

The parameter β determines the shape of the window and thus controls the trade-off between main-lobe widt and side-lobe amplitude. A *Kaiser window* is nearly optimum in the sense of having the most energy in its mai lobe for a given side-lobe amplitude.

There are two empirically derived relationships for the Kaiser window that facilitate the use of these window to design FIR filters. The first relates the stopband ripple of a low-pass filter, $\alpha_s = -20\log(\delta_s)$, to the parameter β

$$\beta = \begin{cases} 0.1102(\alpha_s - 8.7) & \alpha_s > 50 \\ 0.5842(\alpha_s - 21)^{0.4} + 0.07886(\alpha_s - 21) & 21 \le \alpha_s \le 50 \\ 0.0 & \alpha_s < 21 \end{cases}$$

The second relates N to the transition width Δf and the stopband attenuation α_s,

$$N = \frac{\alpha_s - 7.95}{14.36\Delta f} \qquad \alpha_s \ge 21$$

Note that if $\alpha_s < 21$ dB, a rectangular window may be used ($\beta = 0$), and $N = 0.9/\Delta f$.

Example1:

Suppose that we would like to design a low-pass filter with a cutoff frequency $\omega_c = \pi/4$, a transition width $\Delta\omega = 0.02\pi$, and a stopband ripple $\delta_s = 0.01$. Because $\alpha_s = -20\log(0.01) = -40$, the Kaiser window parameter is

$$\beta = 0.5842(40 - 21)^{0.4} + 0.07886(40 - 21) = 3.4$$

With $\Delta f = \Delta\omega/2\pi = 0.01$, we have

$$N = \frac{40 - 7.95}{14.36 \cdot (0.01)} = 224$$

Therefore,

$$h(n) = h_d(n)w(n)$$

where

$$h_d(n) = \frac{\sin[(n - 112)\pi/4]}{(n - 112)\pi}$$

is the unit sample response of the ideal low-pass filter.

Example 2:

Use the window design method to design a linear phase FIR filter of order $N = 24$ to approximate the following ideal frequency response magnitude:

$$|H_d(e^{j\omega})| = \begin{cases} 1 & |\omega| \le 0.2\pi \\ 0 & 0.2\pi < |\omega| \le \pi \end{cases}$$

The ideal filter that we would like to approximate is a low-pass filter with a cutoff frequency $\omega_p = 0.2\pi$. With $N = 24$, the frequency response of the filter that is to be designed has the form

$$H(e^{j\omega}) = \sum_{n=0}^{24} h(n)e^{-jn\omega}$$

Therefore, the delay of $h(n)$ is $\alpha = N/2 = 12$, and the ideal unit sample response that is to be windowed is

$$h_d(n) = \frac{\sin[0.2\pi(n-12)]}{(n-12)\pi}$$

All that is left to do in the design is to select a window. With the length of the window fixed, there is a trade-off between the width of the transition band and the amplitude of the passband and stopband ripple. With a rectangular window, which provides the smallest transition band,

$$\Delta\omega = 2\pi \cdot \frac{0.9}{24} = 0.075\pi$$

and the filter is

$$h(n) = \begin{cases} \dfrac{\sin[0.2\pi(n-12)]}{(n-12)\pi} & 0 \le n \le 24 \\ 0 & \text{otherwise} \end{cases}$$

However, the stopband attenuation is only 21 dB, which is equivalent to a ripple of $\delta_s = 0.089$. With a Hamm window, on the other hand,

$$h(n) = \left[0.54 - 0.46\cos\left(\frac{2\pi n}{24}\right)\right] \cdot \frac{\sin[0.2\pi(n-12)]}{(n-12)\pi} \qquad 0 \le n \le 24$$

and the stopband attenuation is 53 dB, or $\delta_s = 0.0022$. However, the width of the transition band increases

$$\Delta\omega = 2\pi \cdot \frac{3.3}{24} = 0.275\pi$$

which, for most designs, would be too wide.

Frequency Sampling Filter Design

Another method for FIR filter design is the frequency sampling approach. In this approach, the desired frequen response, $H_d(e^{j\omega})$, is first uniformly sampled at N equally spaced points between 0 and 2π:

$$H(k) = H_d\left(e^{j2\pi k/N}\right) \qquad k = 0, 1, \ldots, N-1$$

These frequency samples constitute an N-point DFT, whose inverse is an FIR filter of order $N-1$:

$$h(n) = \frac{1}{N} \sum_{k=0}^{N-1} H(k) e^{j2\pi nk/N} \qquad 0 \le n \le N-1$$

The relationship between $h(n)$ and $h_d(n)$ is

$$h(n) = \sum_{k=-\infty}^{\infty} h_d(n + kN) \qquad 0 \le n \le N-1$$

FINITE WORD-LENGTH EFFECTS

In implementing a discrete-time system in hardware or software, it is important to consider the finite word-lengt effects. For example, if a filter is to be implemented on a fixed-point processor, the filter coefficients must b quantized to a finite number of bits. This will change the frequency response characteristics of the filter. In thi section, we look at the finite precision effects in digital filter implementations.

Binary Representation of Numbers

There are two basic systems for representing numbers in a digital system: fixed point and floating point. Ther is a trade-off in which type of representation to use. The dynamic range that is available in a floating-poi representation is much larger than with fixed-point numbers. However, fixed-point processors are typically fast and less expensive. Below, we briefly describe these number representations.

Fixed Point

In the binary representation of a real number, x, using $B+1$ bits, there are three commonly used formats: sig magnitude, one's complement, and two's complement, with two's complement being the most common. In thes systems, the only difference is in the way that negative numbers are represented.

1. *Sign magnitude*: With a sign-magnitude format, a number x is represented as

$$x = X_m(-1)^{b_0} \cdot \sum_{i=1}^{B} b_i 2^{-i}$$

where X_m is an arbitrary scale factor and where each of the *bits* b_i are either 0 or 1. Thus, b_0 is the sig bit, and the remaining bits represent the magnitude of the fractional number. Bit b_1 is called the *mos significant bit (MSB)*, and b_B is called the *least significant bit* (LSB). For example, with $X_m = 1$,

$$x = 0.8125 = 0.11010$$

and

$$-x = -0.8125 = 1.11010$$

2. *One's complement*: In one's complement form, a negative number is represented by complementin all of the bits in the binary representation of the positive number. For example, with $X_m = 1$ an $x = 0.8125 = 0.11010$,
$$-x = -0.8125 = \overline{0.11010} = 1.00101$$
3. *Two's complement*: With a two's complement format, a real number x is represented as

$$x = X_m\left(-b_0 + \sum_{i=1}^{B} b_i 2^{-i}\right)$$

Thus, negative numbers are formed by complementing the bits of the positive number and adding 1 to th least significant bit. For example, with $X_m = 1$, the two's complement representation of $x = -0.812$ is

$$x = -0.8125 = \overline{0.11010} + 0.00001 = 1.00110$$

Note that with $B + 1$ bits, the smallest difference between two quantized numbers, the *resolution*, is

$$\Delta = X_m 2^{-B}$$

and all quantized numbers lie on the *range* $-X_m \le x < X_m$.

Floating Point

For a word length of $B + 1$ bits in a fixed-point number system, the resolution is constant over the entire range o numbers, and the resolution decreases (Δ increases) in direct proportion to the dynamic range, $2X_m$. A floating point number system covers a larger range of numbers at the expense of an overall decrease in resolution, wit the resolution varying over the entire range of numbers. The representation used for floating-point numbers i typically of the form

$$x = M \cdot 2^E$$

where M, the *mantissa*, is a signed B_M-bit fractional binary number with $\frac{1}{2} \le |M| < 1$, and E, the *exponent*, i a B_E-bit signed integer. Because M is a signed fraction, it may be represented using any of the representation described above for fixed-point numbers.

Quantization Errors in Fixed-Point Number Systems

In performing computations within a fixed- or floating-point digital processor, it is necessary to quantize numbers by either truncation or rounding from some level of precision to a lower level. For example, because multiplying two 16-bit fixed-point numbers will produce a product with up to 31 bits of precision, the product will generally need to be quantized back to 16 bits. Truncation and rounding introduce a quantization error

$$e = Q[x] - x$$

where x is the number to be quantized and $Q[x]$ is the quantized number. The characteristics of the error depend upon the number representation that is used. Truncating numbers that are represented in sign-magnitude form result in a quantization error that is negative for positive numbers and positive for negative numbers. Thus, the quantization error is symmetric about zero and falls in the range

$$-\Delta \leq e \leq \Delta$$

where

$$\Delta = X_m 2^{-B}$$

On the other hand, for a two's complement representation, the truncation error is always negative and falls in the range

$$-\Delta \leq e \leq 0$$

With rounding, the quantization error is independent of the type of fixed-point representation and falls in the range

$$-\frac{\Delta}{2} \leq e \leq \frac{\Delta}{2}$$

For floating-point numbers, the mantissa is either rounded or truncated, and the size of the error depends on the value of the exponent.

Quantization of Filter Coefficients

In order to implement a filter on a digital processor, the filter coefficients must be converted into binary form. This conversion leads to movements in the pole and zero locations and a change in the frequency response of the filter. The accuracy with which the filter coefficients can be specified depends upon the word length of the processor, and the sensitivity of the filter to coefficient quantization depends on the structure of the filter, as well as on the locations of the poles and zeros.

For an FIR filter,

$$H(z) = \sum_{n=0}^{N} h(n) z^{-n}$$

when the coefficients are quantized, the system function becomes

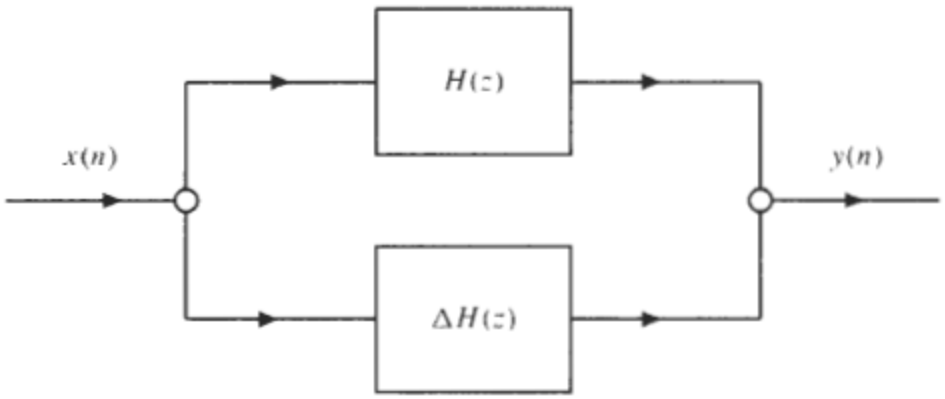
$$\hat{H}(z) = \sum_{n=0}^{N} \hat{h}(n) z^{-n} = \sum_{n=0}^{N} [h(n) + \Delta h(n)] z^{-n} = H(z) + \Delta H(z)$$

Thus, the quantization errors may be modeled as $H(z)$ in parallel with $\Delta H(z)$ as shown in Fig. . If we assume that the coefficients $h(n)$ are less than 1 in magnitude, and that the coefficients are rounded to $B + 1$ bits,

$$-2^{-(B+1)} < \Delta h(n) < 2^{-(B+1)}$$

Therefore, a loose bound on the error in the frequency response is

$$|\Delta H(e^{j\omega})| = \left| \sum_{n=0}^{N} \Delta h(n)\, e^{-jn\omega} \right| \leq \sum_{n=0}^{N} |\Delta h(n)| \leq (N + 1) 2^{-(B+1)}$$



Model for the coefficient quantization error in FIR filters.

As with IIR filters, if the zeros are tightly clustered, the zero locations will be sensitive to coefficien quantization errors. However, FIR filters are commonly implemented in direct form for two reasons:

1. The zeros of FIR filters are not generally tightly clustered.
2. In direct form, linear phase is easily preserved.

Round-Off Noise

Round-off noise is introduced into a digital filter when products or sums of products are quantized. For example if two $(B + 1)$-bit numbers are multiplied, the product is a $(2B + 1)$-bit number. If the product is to be saved i a $(B + 1)$-bit register or used in a $(B + 1)$-bit adder, it must be quantized to $(B + 1)$-bits, which results in th addition of *round-off noise*. This round-off noise propagates through the filter and appears at the output of th filter as round-off noise. In this section, we illustrate the analysis of round-off noise effects by example.

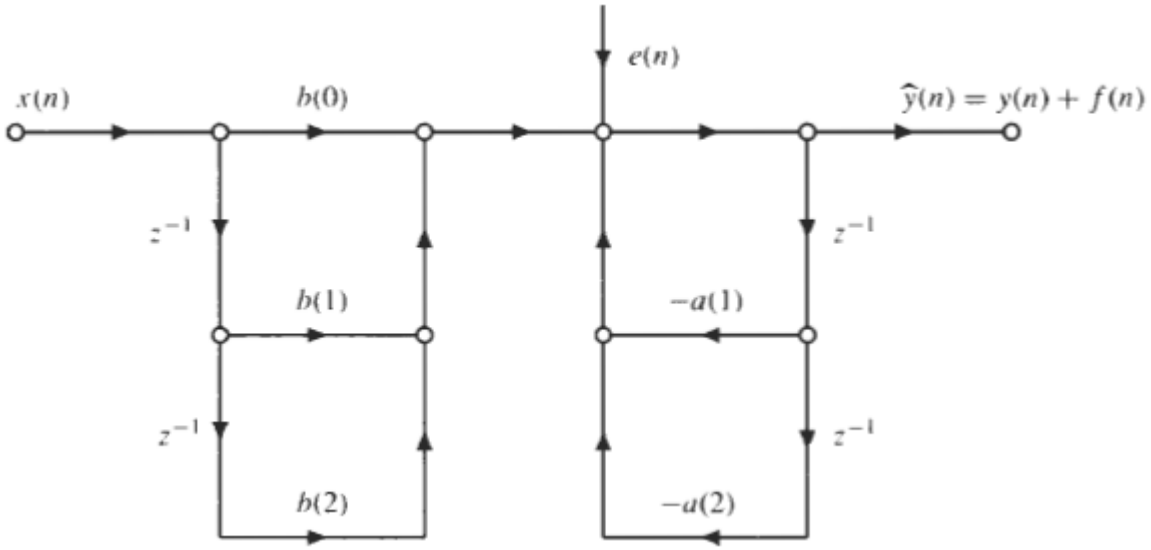
Example:

Consider the first-order all-pole filter with a system function

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

If the input to this filter, $e(n)$, is zero mean white noise with a variance σ_e^2, the variance of the output will b

$$\sigma_f^2 = \sigma_e^2 \sum_{n=-\infty}^{\infty} |h(n)|^2 = \sigma_e^2 \sum_{n=0}^{\infty} |\alpha|^{2n} = \sigma_e^2 \frac{1}{1 - |\alpha|^2}$$



Thus, the quantization noise is filtered only by the poles of the filter, and the output noise satisfies the differenc equation

$$f(n) = e_a(n) - \sum_{k=1}^{2} a(k) f(n-k)$$

If the noise sources are uncorrelated, as assumed by the third property above, the variance of $e_a(n)$ is the sum
the variances of the five noise sources, or

$$\sigma_a^2 = 5\sigma_c^2 = 5 \cdot \frac{2^{-2B}}{12}$$

Assuming that the filter is stable, and that the poles of the filter are complex,

$$\frac{1}{1 + a(1)z^{-1} + a(2)z^{-2}} = \frac{1}{(1 - re^{j\theta}z^{-1})(1 - re^{-j\theta}z^{-1})}$$

the variance of the output noise is

$$\sigma_f^2 = 5\frac{2^{-2B}}{12}\frac{1}{2\pi j}\oint_C \frac{1}{A(z)A(z^{-1})}z^{-1}\,dz$$

$$= 5\frac{2^{-2B}}{12}\frac{1}{2\pi j}\oint_C \frac{z\,dz}{(z - re^{j\theta})(z - re^{-j\theta})(1 - re^{j\theta}z)(1 - re^{-j\theta}z)}$$

Using Cauchy's residue theorem to evaluate this integral, we find that

$$\sigma_f^2 = 5\frac{2^{-2B}}{12}\left(\frac{1 + r^2}{1 - r^2}\right)\frac{1}{r^4 + 1 - 2r^2\cos 2\theta}$$

Note that as the poles move closer to the unit circle, $r \to 1$, the variance of the output noise increases.

Overflow

Another issue in fixed-point implementations of discrete-time systems is *overflow*. If each fixed-point numb
is taken to be a fraction that is less than 1 in magnitude, each node variable in the network should be constrain
to be less than 1 in magnitude in order to avoid overflow. If we let $h_k(n)$ denote the unit sample response of t
system relating the input $x(n)$ to the kth node variable, $w_k(n)$,

$$|w_k(n)| = \left|\sum_{m=-\infty}^{\infty} x(n - m)h_k(m)\right| \le X_{\max}\sum_{m=-\infty}^{\infty}|h_k(m)|$$

where $X_{\max}$ is the maximum value of the input $x(n)$. Therefore, a sufficient condition that $|w_k(n)| < 1$ so th no overflow occurs in the network is

$$X_{\max} \leq \frac{1}{\sum_{m=-\infty}^{\infty} |h_k(m)|}$$

for all nodes in the network. If this is not satisfied, $x(n)$ may be scaled by a factor s so that

$$sX_{\max} \leq \frac{1}{\sum_{m=-\infty}^{\infty} |h_k(m)|}$$

UNIT – V

Multirate signal processing:

Lets say we have $x(n) = x_a(nT)$ with a sampling rate $F_s = \frac{1}{T}$ and want to reduce its sampling rate by an integer factor D. That is: $x_d(n) = x(nD)$ *or* $x_d(n) = x_a(nT') = x_a(n.D.T.) = x(nD)$ where $T'=D.T$ the decimated sampling rate.

$$X(\omega) = FFT\{x(n)\} = \frac{1}{T}\sum_{k=-\infty}^{+\infty} X_a\left(\frac{\omega}{T} - \frac{2\pi k}{T}\right) \qquad (1), \text{ and}$$

$$X_d(\omega) = \frac{1}{D\cdot T}\sum_{\ell=-\infty}^{+\infty} X_a\left(\frac{\omega}{D\cdot T} - \frac{2\pi \ell}{D\cdot T}\right) \text{ as } T' = D.T \qquad (2)$$

Comparing 1 and 2 $\rightarrow X_d(\omega_x) = \frac{1}{D} X\left(\frac{\omega_y}{D}\right) \omega_y = D\cdot\omega_x$.

If the original signal was over sampled and $BW_x \le \frac{2\pi}{D}$, then there is no aliasing. However, if this is not the case, then to avoid aliasing, one should filter *x(n)* by $\frac{\pi}{D}$ prior to down sampling by a factor *D*.

UP Sampling by a Factor L

In this case, $T' = \frac{T}{L}$, therefore, $x_u(n) = x_a(nT') = x_a\left(\frac{nT}{L}\right) = x\left(\frac{n}{L}\right)$

$$x_u = \begin{cases} x\left(\frac{n}{L}\right) & n = 0,\pm L,\pm 2L,\ldots = \sum_{-\infty}^{+\infty} x(k)\delta(n-kL) \\ 0 & else \end{cases}$$

Now in order to interpolate between the stretched samples of *x(n)*, we must pass the $x_u(n)$ through a LPF with cutoff frequency $\omega_c = \frac{\pi}{L}$.

$$X_u(\omega_x) = \sum_{n=-\infty}^{+\infty}\left[\sum_{K=-\infty}^{+\infty} x(k)\delta(n-kL)\right]e^{-j\omega n}$$

$$= \sum_{k=-\infty}^{+\infty} x(k)e^{-j\omega Lk} = X(\omega_y L) \Rightarrow \omega_y = \frac{\omega_x}{L}$$

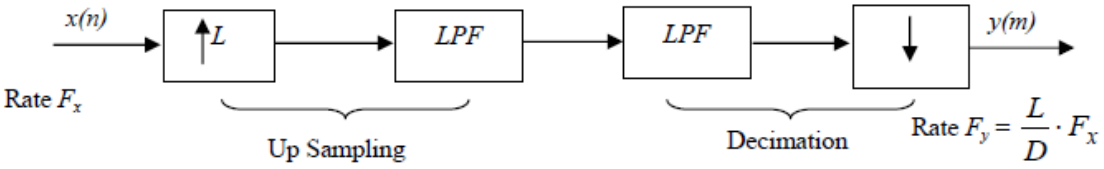
So $X(\omega)$ becomes compressed for $L > 0$ and now pass $X_u(\omega)$ through LPF:

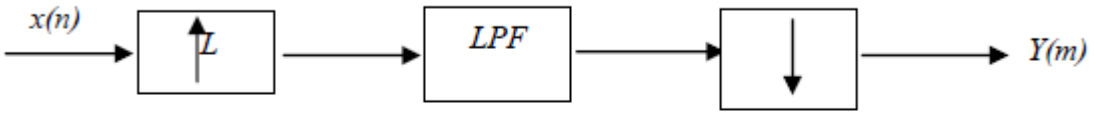
$$H(\omega_y) = \begin{cases} C & 0 \le |\omega_y| \le \pi/L \\ 0 & else \end{cases}$$

Therefore, $Y(\omega_y) = \begin{cases} CX(\omega_y L) & 0 \le |\omega_y| \le \dfrac{\pi}{L} \\ 0 & else \end{cases}$

Changing the Sampling Rate by a Factor $\frac{L}{D}$

We should first up sample by L, the down sample by D.

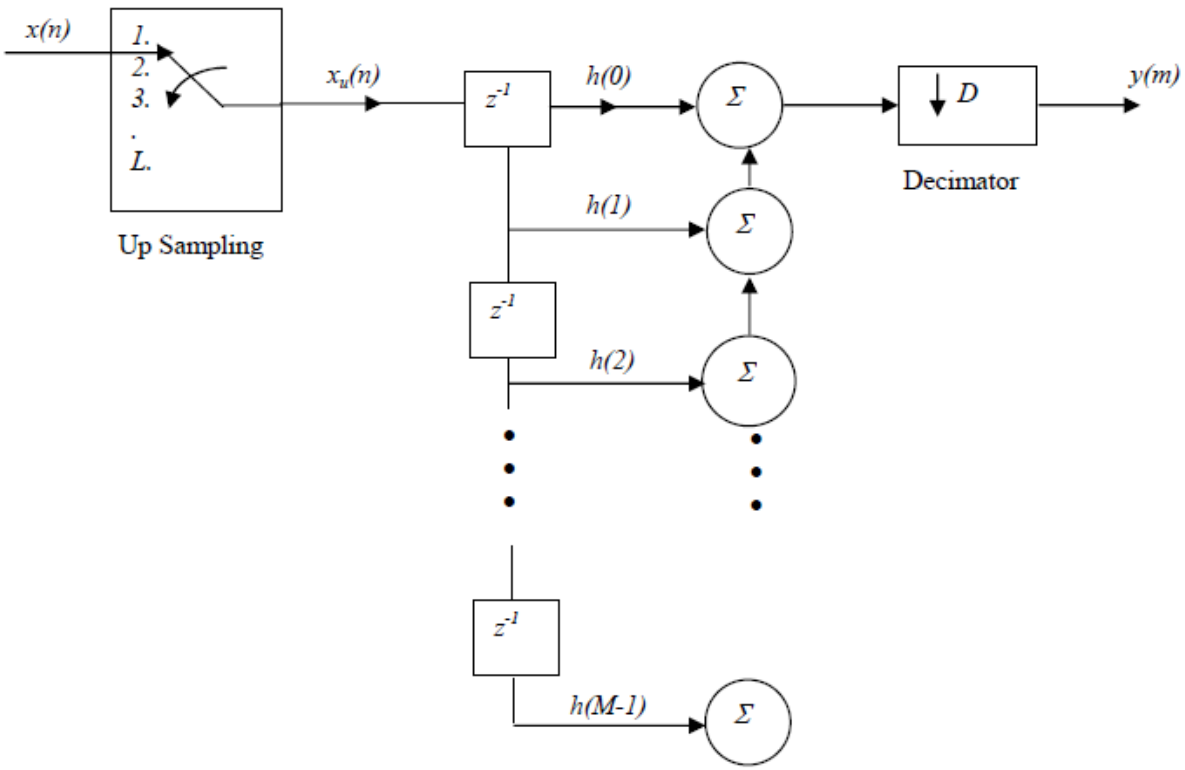




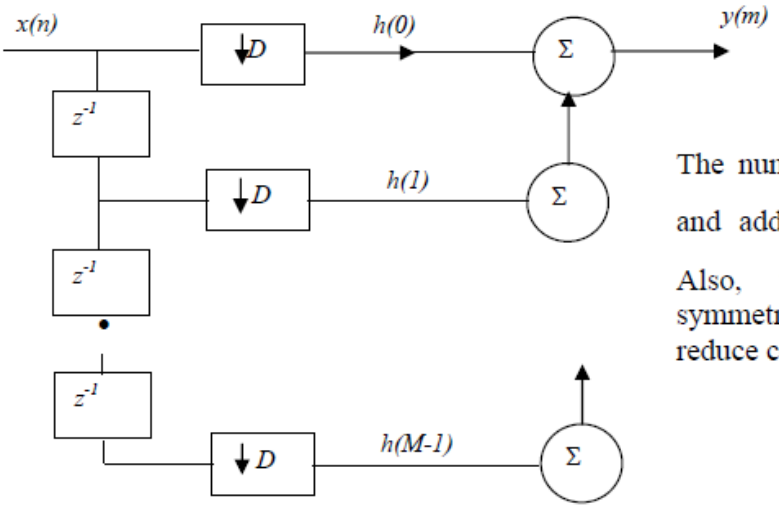
$$Y(\omega_y) = \begin{cases} \frac{L}{D} X\left(\frac{\omega_y}{D}\right) & 0 \le |\omega_y| \le \min\left(\pi, \frac{\pi D}{L}\right) \\ 0 & else \end{cases}$$

Filter Design for Sampling Rate Conversion

Lets use FIR filter $H(z) = \sum_{k=0}^{M-1} h(k) z^{-k}$



This filter is simple but very inefficient because $(L-1)$ times, the $x_u(n)$ is zero and still the filter processes it. If L is large, then most of the inputs of the LPF filter is zero. Also, the output only one of every D samples is needed. If we do this instead for a decimator:



The number of multiplication and addition is at $\frac{F_x}{D}$ rate. Also, we can use the symmetry of $h(k)$ and still reduce computation.

Sampling Rate Conversion by an Arbitrary Number

Some time the rate $\frac{L}{D}$, L is very large, i.e., $\frac{L}{D}=\frac{1023}{511}$ meaning to 1023 subfilters or polyphase filters so seeking other methods.

1st Order Approximation

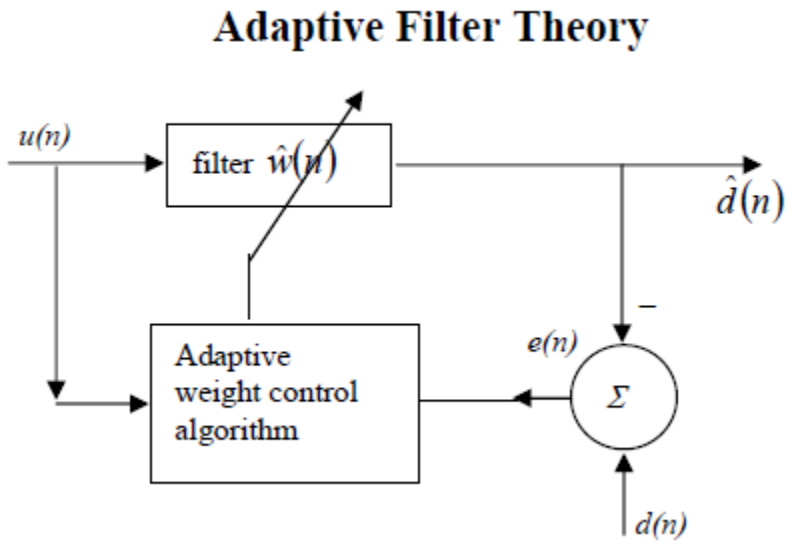
Let's assume T_x is the sample interval for $x(n)$ and we want to upsample it by r ($T=\frac{T_x}{r}$). We can express $\frac{1}{r}$ as: $\frac{1}{r}=\frac{K}{L}+\beta$, where K and L are two integers and $0<\beta<\frac{1}{L}\Rightarrow\frac{K}{L}<\frac{1}{r}<\frac{K+1}{L}$. L is infact the number of polyphase filters due to upsampling.

For example, let $r = 2.2$ then $\frac{K}{L} = \frac{2}{6} < \frac{1}{2.2} < \frac{3}{6} = \frac{K+1}{L}$. So the number of filters for upsampling is 6. The time spacing between samples of the interpolated sequence is $\frac{T_x}{L}$. However, the desired conversion rate $r = 2.2$, for $L = 6$ corresponds to a decimation factor 2.727, which falls between $K = 2$ and $K = 3$. In the first appoximation method, we achieve the desired decimation rate by selecting the output sample from the filter closest in time to the desired sampling rate.

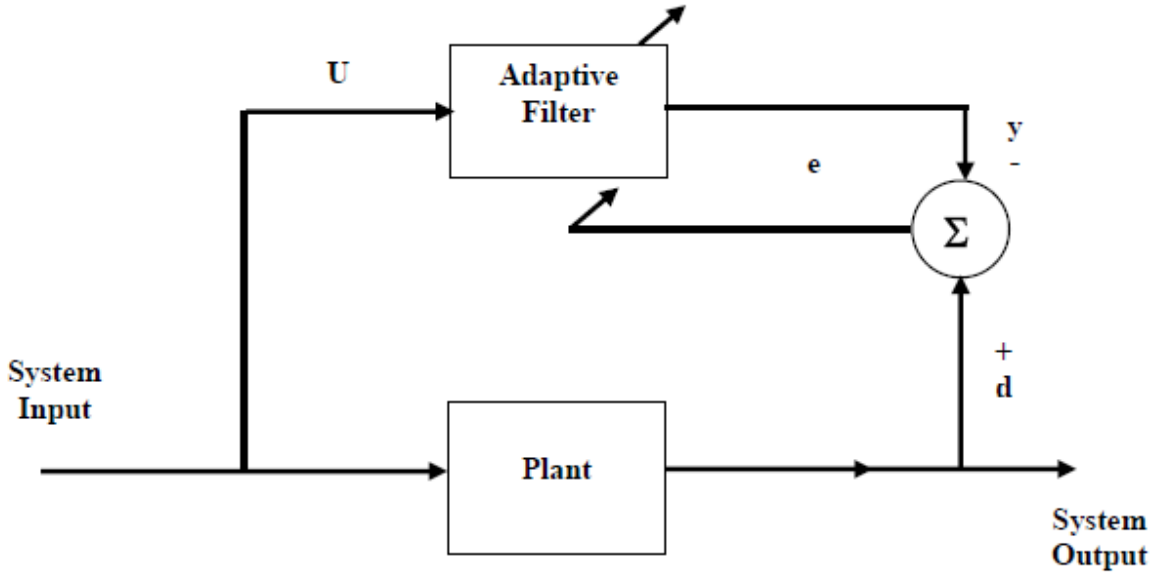
Note: upsampled data idealy has to be downsampled to 2.7. So we select either 2 or 3, anyone closer to the ideal one.

Adaptive Filter:

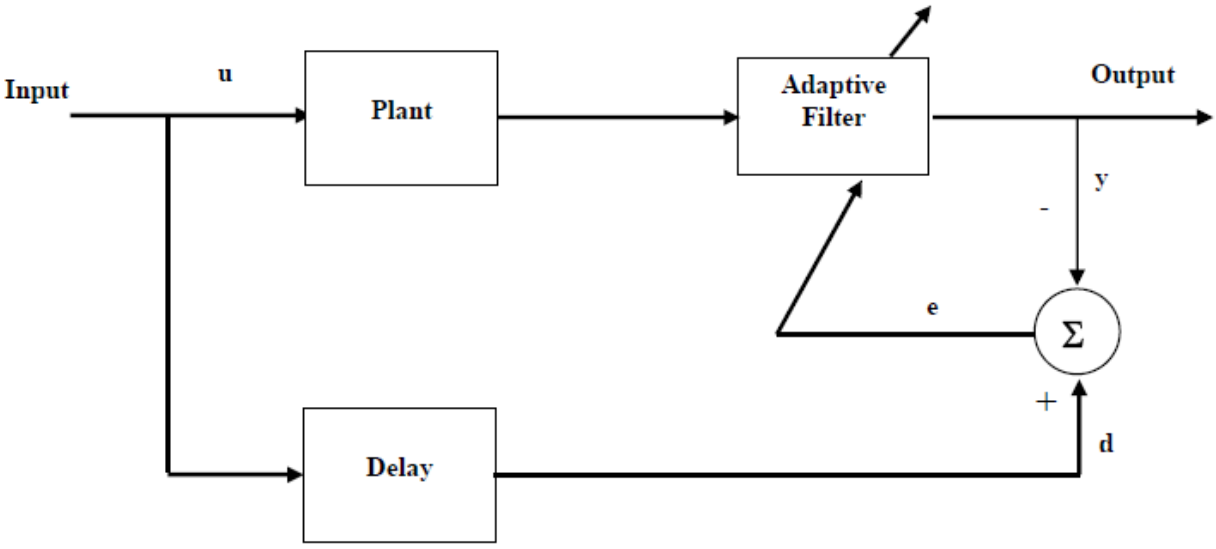
Linear Filtering will be optimal only if it is designed with some knowledge about the input data.If this information is not known,then adaptive filters are used.The adjustable parameters in the filter are assigned with values based on the estimated statistical nature of the signals.So,these filters are adaptable to the changing environment.Adaptive filtering finds its application in adaptive noise cancelling,line enhancing,frequency tracking,channel equalizations, etc.



Few applications of Adaptive filter are:

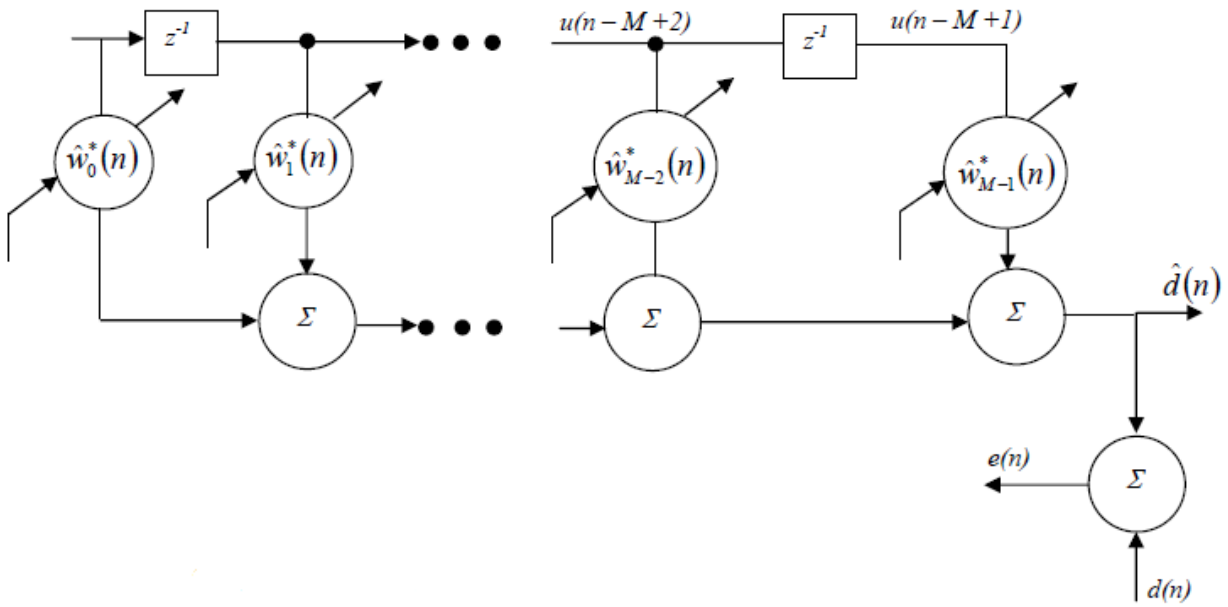


a) Identification



b) Inverse Modelling

Least-Mean-Square (LMS) Adaptation Algorithm:



Parameters: M = # of taps

μ = step-size parameter

$$o < \mu < \frac{2}{tap-input\ \ power} = \frac{2}{\sum_{k=0}^{M-1} E\left[|u(n-k)|^2\right]}$$

Initialization: $\underline{\hat{w}}(0) = \underline{0}$ unless there is a prior knowledge.

Given: $\underline{u}(n) = M$ - by -1 tap input vector at time n.

$d(n)$ = desired response time at n

To be computed:

$\underline{\hat{w}}(n+1)$ = estimate of tap-weight vector at time $n + 1$

Computation:

$$e(n) = d(n) - \underline{\hat{w}}^H(n) \cdot \underline{u}(n)$$

$$\underline{\hat{w}}(n+1) = \underline{\hat{w}}(n) + \mu \underline{u}(n) \cdot e^*(n)$$

Normalized algorithm:

M = # of taps

$\hat{\mu}$ = adaptation constant, $o < \hat{\mu} < 2$

a = a small positive constant

Initialization: $\underline{\hat{w}}(0) = \underline{0}$

Given: *u(n)*: *M* by 1 tap-input vector at time *n*

d(n): desired response at time *n*

Compute:

$$e(n) = d(n) - \underline{\hat{w}}^H(n)\,\underline{u}(n)$$

$$\underline{\hat{w}}(n+1) = \underline{\hat{w}}(n) + \frac{\hat{\mu}}{a + \left\|\underline{u}(n)\right\|^2} \cdot \underline{u}(n)e^*(n)$$

As you see, $\hat{\mu}$ is divided by the norm of *u(n)* and hence, the name "Normalized". In case of having very small input, numerical difficulties may arise due to the division to $\| \underline{u}(n)\|^2$, then we may define $\mu(n) = \frac{\hat{\mu}}{\left\|\underline{u}(n)\right\|^2}$ and in this light Normalized LMS may be viewed as LMS with a time-varying step-size parameter. The rate of convergence of the Normalized LMS is faster than the conventional LMS Algorithm.

Recursive Least Squares Algorithm (RLS)

M = number of taps and λ = forgetting factor ≤ 1

Given $u(n)$: M-by-1 tap-input vector

$d(n)$: desired response.

Initialize: $\underline{P}(0) = \sigma^{-1} \cdot I$, σ = small positive constant (i.e. 0.25)

$\underline{\hat{w}}(0) = \underline{0}$

For each instant of time, $n = 1, 2, \ldots$, compute:

$$\underline{K}(n) = \frac{\lambda^{-1}\underline{P}(n-1)\underline{u}(n)}{1+\lambda^{-1}\underline{u}^{H}(n)\cdot\underline{P}(n-1)\underline{u}(n)}$$

$$e(n) = d(n) - \underline{\hat{w}}^{H}(n-1)\cdot\underline{u}(n)$$

$$\underline{\hat{w}}(n) = \underline{\hat{w}}(n-1) + \underline{K}(n)e^{*}(n)$$

$$\underline{P}(n) = \lambda^{-1}\underline{P}(n-1) - \lambda^{-1}\underline{K}(n)\cdot\underline{u}^{H}(n)\cdot\underline{P}(n-1)$$

A few key features of the RLS Algorithm

- The mean of the learning curve of the RLS algorithm converges in about $2M$ iterations, where M is the number of taps of the filter. Therefore, the RLS convergence rate is much faster than that of LMS Algorithm.
- As the number of iterations, n, approaches infinity, the mean-squared error of RLS approaches a final value equal to the variance σ^2 of the measurement error. In other words, in theory, RLS algorithm produce zero error as $n \to \infty$.
- Convergence of the RLS algorithm in the mean-squared sense is independent of the eigen value spread of the correlation matrix of the input vector.

Speech Compression:

A voice signal has the frequency range of 300 to 3000Hz.It is sampled at a rate of 8 kHz and the word length of the digitized signal is 12 bits.

The redundancy present in the voice signals can be reduced by signal compression and coding.

The different voice compression and coding techniques are:

i)waveform coding-non-uniform,differential,adaptive quantization.

ii)Transform coding-transform the voice signal to an orthogonal system and then coding the transform.

Iii)Frequency band encoding-frequency range of voice signals are divided into discrete channels and then each channel is coded separately.

iv)Parametric methods-linear prediction.

Linear predictive coding(LPC):

LPC works on the principle of analysis by synthesis which means that the encoder includes a replica of the decoder in its design.DSP circuits can be used to implement LPC.Using LPC,high levels of compression can be achieved.It involves the following operating steps:

1.Sampling the input speech at a designed rate using rectifier and 20Hz low pass filter.

2.Analyzing the block of audio waveform samples.

3.Determining the perceptual features of the audio waveform.feature extraction is done by analyzing the block of digitized samples known as segment.The perceptual features are:
pitch
period
loudness
vocal tract excitation parameters.

4.Quantizing the sampled segments.

5.Computing the error signal which is the difference between the original speech and its regenerated version.

6.Determining the new set of filter coefficients[c1,c2…cn] using the current set of vocal tract model coefficients and linear combination of the previous set of coefficients to minimize the error.

7.Transmitting the new set of coefficients for each quantized segment,in a string of frames.Frame contains fields for pitch,loudness,period and whether the signal is voiced or unvoiced and a new set of computed filter coefficients.