INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad -500 043

ELECTRONICS AND COMMUNICATION ENGINEERING

COURSE LECTURE NOTES

Course Name	**DIGITAL SIGNAL PROCESSING**
Course Code	AEC012
Programme	B.Tech
Semester	VI
Course Coordinator	Dr. S. China Venkateswarlu, Professor
Course Faculty	Dr. G Manisha, Associate Professor Ms. S Sushma, Assistant Professor Mr. K Chaitanya, Assistant Professor
Lecture Numbers	1-60
Topic Covered	All

COURSE OBJECTIVES (COs):

The course should enable the students to:	
I	Provide background and fundamental material for the analysis and processing of digital signals and to familiarize the relationships between continuous-time and discrete-time signals and systems.
II	Study fundamentals of time, frequency and z-plane analysis and to discuss the inter-relationships of these analytic method and to study the designs and structures of digital (IIR and FIR) filters from analysis to synthesis for a given specifications.
III	Introduce a few real-world signal processing applications.
IV	Acquainting FFT algorithm, multi-rate signal processing techniques and finite word length effects.

COURSE LEARNING OUTCOMES (CLOs):

Students, who complete the course, will have demonstrated the ability to do the following:

AEC012.01	Understand how digital to analog (D/A) and analog to digital (A/D) converters operate on a signal and be able to model these operations mathematically.
AEC012.02	Define simple non-periodic discrete-time sequences such as the impulse and unit step, and perform time shifting and time-reversal operations on such sequences.
AEC012.03	Given the difference equation of a discrete-time system to demonstrate linearity, time-invariance, causality and stability, and hence show whether or not a given system belongs to the important class of causal, LTI systems.
AEC012.04	Given the impulse response of a causal LTI system, show whether or not the system is bounded-input/bounded-output (BIBO) stable.
AEC012.05	Perform time, frequency and Z-transform analysis on signals.
AEC012.06	From a linear difference equation of a causal LTI system, draw the Direct Form I and Direct Form II filter realizations.
AEC012.07	Knowing the poles and zeros of a transfer function, make a rough sketch of the gain response.
AEC012.08	Define the Discrete Fourier Transform (DFT) and the inverse DFT (IDFT) of length N.
AEC012.09	Understand the inter-relationship between DFT and various transforms.
AEC012.10	Understand the significance of various filter structures and effects of round-off errors.
AEC012.11	Understand the fast computation of DFT and appreciate the FFT Processing.
AEC012.12	Design of infinite impulse response (IIR) filters for a given specification.
AEC012.13	Design of finite impulse response (FIR) filters for a given specification.
AEC012.14	Compare the characteristics of IIR and FIR filters.
AEC012.15	Understand the tradeoffs between normal and multi rate DSP techniques and finite length word effects.
AEC012.16	Understand the signal interpolation and decimation, and explain their operation
AEC012.17	Explain the cause of limit cycles in the implementation of IIR filters.

SYLLABUS

UNIT-I	**REVIEW OF DISCRETE TIME SIGNALS AND SYSTEMS:**	**Classes: 10**
Discrete time signal definition; Signal classification; Elementary signals; Transformation of elementary signals; Concept of digital frequency; Discrete time system definition; System classification; Linear time invariant (LTI) system; Properties of the LTI system; Time domain analysis of discrete time systems; Impulse response; The convolution sum; Methods of evaluating the convolution sum; Filtering using overlap-save and overlap-add method; Realization of digital filters: Concept of IIR and FIR filters; Realization structures for IIR and FIR filters using direct form-I and direct form-II, cascade, lattice and parallel.		
UNIT -II	**DISCRETE FOURIER TRANSFORM AND EFFICIENT COMPUTATION:**	**Classes: 09**
Introduction to discrete time Fourier transform (DTFT); Discrete Fourier transform (DFT) definition; Properties of DFT; Linear and circular convolution using DFT; Fast-Fourier-transform (FFT): Direct		

computation of DFT; Need for efficient computation of the DFT (FFT algorithms); Radix-2 FFT algorithm for the computation of DFT and IDFT using decimation-in-time and decimation-in-frequency algorithms; General Radix-N FFT.

UNIT-III	STRUCUTRE OF IIR FILTERS:	Classes: 09

Analog filters: Butterworth filters; Chebyshev type-1 & type-2 filters; Analog transformation of prototype LPF to HPF/BPF/BSF.

Transformation of analog filters into equivalent digital filters using impulse invariant method and bilinear transform method; Matlab programs of IIR filters.

UNIT-IV	SYMMETRIC AND ANTISYMMETRIC FIR FILTERS:	Classes: 09

Design of linear phase FIR filters windowing and frequency sampling methods; Equiripple linear phase FIR filters; Parks-McClellan algorithm and remez algorithm; Least-mean-square error filter design; Design of FIR differentiators; Matlab programs of FIR filters; Comparison of FIR & IIR.

UNIT-V	APPLICATIONS OF DSP:	Classes: 10

Multirate signal processing; Decimation; Interpolation; Polyphase structures for decimation and interpolation filters; Structures for rational sampling rate conversion; Applications of multirate signal processing for design of phase shifters, interfacing of digital systems with different sampling rates, sub band coding of speech signals. Analysis of finite word length effects: Representation of numbers; ADC quantization noise, coefficient quantization error, product quantization error, truncation & rounding errors; Limit cycle due to product round-off error; Round-off noise power; Limit cycle oscillations due to overflow in digital filters; Principle of scaling; Dead band effects.

Text Books:

1. John G. Proakis, Dimitris G. Manolakis, Digital signal processing, Principles, Algorithms and Applications, Prentice Hall, 4th Edition, 2007
2. Sanjit K Mitra, Digital signal processing, A computer base approach, McGraw-Hill Higher Education, 4th Edition, 2011.
3. Emmanuel C, Ifeacher, Barrie. W. Jervis, DSP-A Practical Approach, Pearson Education, 2nd Edition, 2002.
4. A.V. Oppenheim, R.W. Schaffer, Discrete Time Signal Processing, PHI, 2nd Edition, 2006.

Reference Books:

1. Li tan, Digital signal processing: fundamentals and applications, Elsevier Science &. Technology Books, 2nd Edition, 2008.
2. Robert J.schilling, Sandra. L.harris, Fundamentals of Digital signal processing using Matlab, Thomson Engineering, 2nd Edition, 2005.
3. Salivahanan, Vallavaraj, Gnanapriya, Digital signal processing‖, McGraw-Hill Higher Education, 2nd Edition, 2009.

UNIT-I

REVIEW OF DISCRETE TIME SIGNALS AND SYSTEMS

Signals-Definition

Anything that carries information can be called as signal. It can also be defined as a physical quantity that varies with time, temperature, pressure or with any independent variables such as speech signal or video signal.

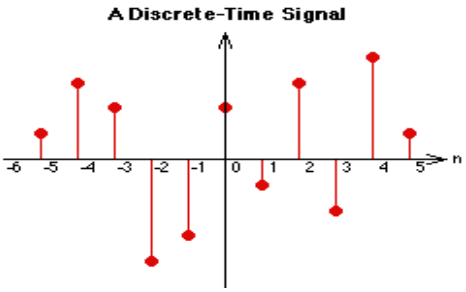
The process of operation in which the characteristics of a signal (Amplitude, shape, phase, frequency, etc.) undergoes a change is known as signal processing.

Note − Any unwanted signal interfering with the main signal is termed as noise. So, noise is also a signal but unwanted.

Discrete Time signals

The signals, which are defined at discrete times are known as discrete signals. Therefore, every independent variable has distinct value. Thus, they are represented as sequence of numbers.

Although speech and video signals have the privilege to be represented in both continuous and discrete time format; under certain circumstances, they are identical. Amplitudes also show discrete characteristics. Perfect example of this is a digital signal; whose amplitude and time both are discrete.



The figure above depicts a discrete signal's discrete amplitude characteristic over a period of time. Mathematically, these types of signals can be formularized as;

$x=\{x[n]\}, -\infty<n<\infty$

Where, n is an integer.

It is a sequence of numbers x, where nth number in the sequence is represented as x[n].

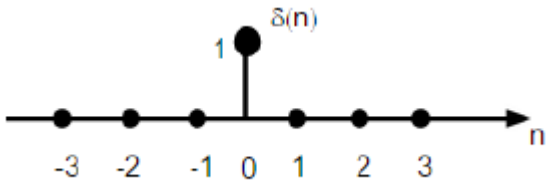
Basic DT Signals

Let us see how the basic signals can be represented in Discrete Time Domain.

Unit Impulse Sequence

It is denoted as δ(n) in discrete time domain and can be defined as;

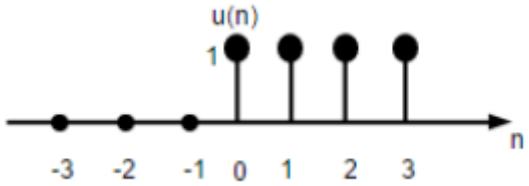
$$\delta(n) = \begin{cases} 1, & for \quad n = 0 \\ 0, & Otherwise \end{cases}$$



Unit Step Signal

Discrete time unit step signal is defined as;

$$U(n) = \begin{cases} 1, & for \quad n \geq 0 \\ 0, & for \quad n < 0 \end{cases}$$

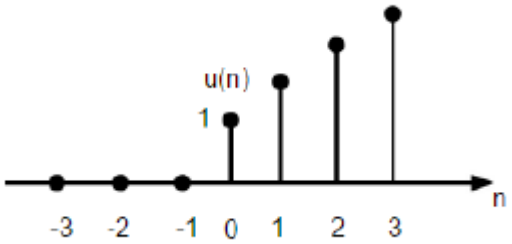


The figure above shows the graphical representation of a discrete step function.

Unit Ramp Function

A discrete unit ramp function can be defined as –

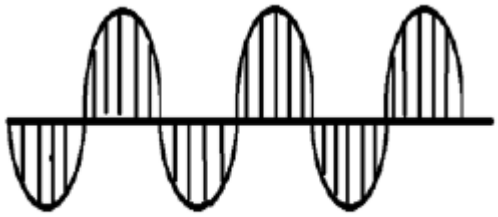
$$r(n) = \begin{cases} n, & for \quad n \geq 0 \\ 0, & for \quad n < 0 \end{cases}$$



The figure given above shows the graphical representation of a discrete ramp signal.

Sinusoidal Signal

All continuous-time signals are periodic. The discrete-time sinusoidal sequences may or may not be periodic. They depend on the value of ω. For a discrete time signal to be periodic, the angular frequency ω must be a rational multiple of 2π.



Discrete sinusoidal signal

A discrete sinusoidal signal is shown in the figure above.

Discrete form of a sinusoidal signal can be represented in the format −

$$x(n)=A\sin(\omega n+\varphi)$$

Here A,ω and ϕ have their usual meaning and n is the integer. Time period of the discrete sinusoidal signal is given by −

$$N=\frac{2\pi m}{\omega}$$

Where, N and m are integers.

Classification of DT Signals

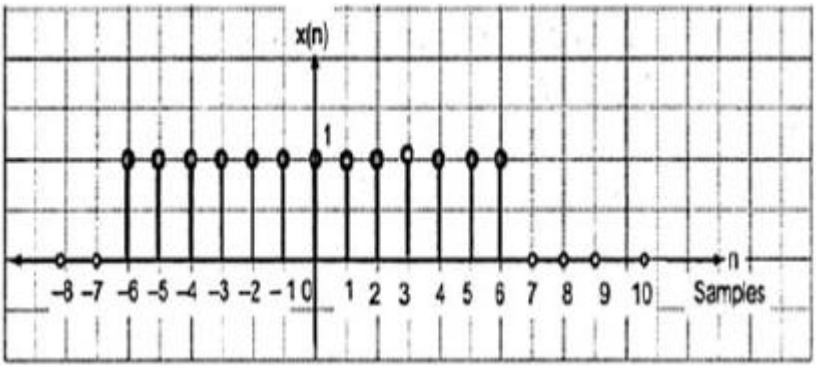
Discrete time signals can be classified according to the conditions or operations on the signals.

Even and Odd Signals

Even Signal

A signal is said to be even or symmetric if it satisfies the following condition;

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

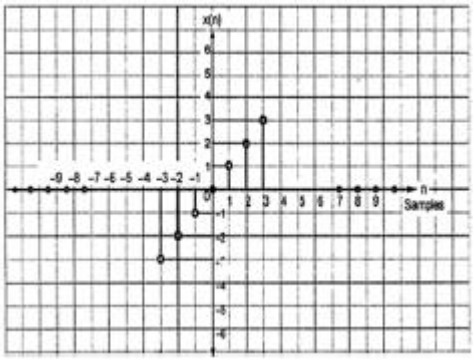


Here, we can see that x(-1) = x(1), x(-2) = x(2) and x(-n) = x(n). Thus, it is an even signal.

Odd Signal

A signal is said to be odd if it satisfies the following condition;

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



From the figure, we can see that x(1) = -x(-1), x(2) = -x(2) and x(n) = -x(-n). Hence, it is an odd as well as anti-symmetric signal.

Periodic and Non-Periodic Signals

A discrete time signal is periodic if and only if, it satisfies the following condition −

$$x(n+N)=x(n)$$

Here, x(n) signal repeats itself after N period. This can be best understood by considering a cosine signal −

$$x(n)=A\cos(2\pi f_0 n+\theta)$$

$$x(n+N)=A\cos(2\pi f_0(n+N)+\theta)=A\cos(2\pi f_0 n+2\pi f0N+\theta)$$

For the signal to become periodic, following condition should be satisfied;

$$x(n+N)=x(n)$$

$$\Rightarrow A\cos(2\pi f_0 n+2\pi f_0 N+\theta)=A\cos(2\pi f_0 n+\theta)$$

i.e. $2\pi f_0 N$ is an integral multiple of 2π

$$2\pi f_0 N=2\pi K$$

$$\Rightarrow \quad N=K/f_0$$

Frequencies of discrete sinusoidal signals are separated by integral multiple of 2π.

Energy and Power Signals

Energy Signal

Energy of a discrete time signal is denoted as E. Mathematically, it can be written as;

$$E=\sum_{n=-\infty}^{+\infty}|x(n)|^2$$

If each individual values of x(n) are squared and added, we get the energy signal. Here x(n) is the energy signal and its energy is finite over time i.e $0<E<\infty$

Power Signal

Average power of a discrete signal is represented as P. Mathematically, this can be written as;

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

Here, power is finite i.e. $0<P<\infty$. However, there are some signals, which belong to neither energy nor power type signal.

Operations on Signals

The basic signal operations which manipulate the signal characteristics by acting on the independent variable(s) which are used to represent them. This means that instead of performing operations like addition, subtraction, and multiplication between signals, we will perform them on the independent variable. In our case, this variable is time (t).

1. Time Shifting

Suppose that we have a signal x(n) and we define a new signal by adding/subtracting a finite time value to/from it. We now have a new signal, y(n). The mathematical expression for this would be x(n± n0).

Graphically, this kind of signal operation results in a positive or negative "shift" of the signal along its time axis. However, note that while doing so, none of its characteristics are altered. This means that the time-shifting operation results in the change of just the positioning of the signal without affecting its amplitude or span.

Let's consider the examples of the signals in the following figures in order to gain better insight into the above information.

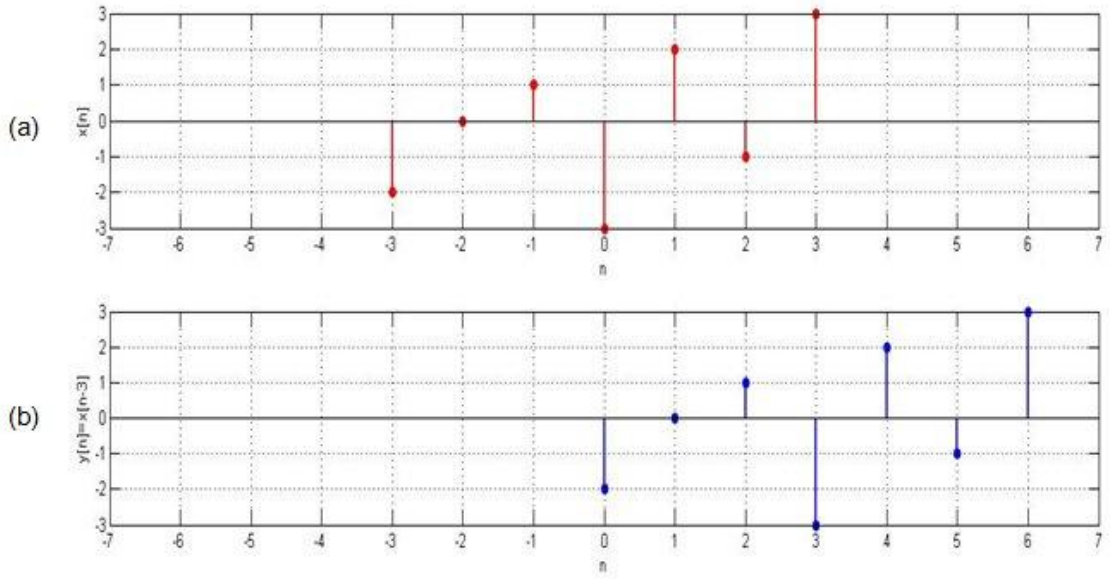


Figure 1. Original signal and its time-delayed version

Here the original signal, x[n], spans from n = -3 to n = 3 and has the values -2, 0, 1, -3, 2, -1, and 3, as shown in Figure 1(a).

Time-Delayed Signals

Suppose that we want to move this signal right by three units (i.e., we want a new signal whose amplitudes are the same but are shifted right three times).

This means that we desire our output signal y[n] to span from n = 0to n = 6. Such a signal is shown as Figure 1(b) and can be mathematically written as y[n] = x[n-3].

This kind of signal is referred to as time-delayed because we have made the signal arrive three units late.

Time-Advanced Signals

On the other hand, let's say that we want the same signal to arrive early. Consider a case where we want our output signal to be advanced by, say, two units. This objective can be accomplished by shifting the signal to the left by two time units, i.e., y[n] = x[n+2].

The corresponding input and output signals are shown in Figure 2(a) and 2(b), respectively. Our output signal has the same values as the original signal but spans from n = -5 to n = 1 instead of n = -3 to n = 3. The signal shown in Figure 2(b) is aptly referred to as a time-advanced signal.

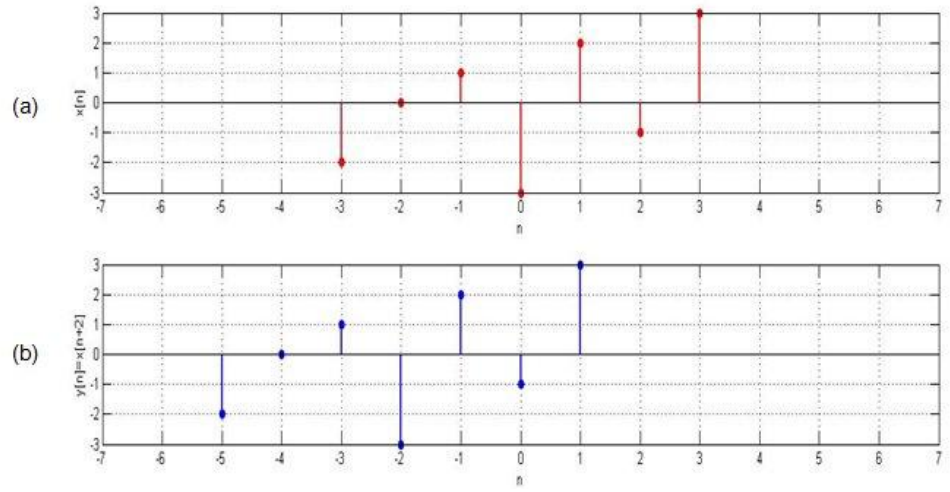


Figure 2. Original signal and its time-advanced version

For both of the above examples, note that the time-shifting operation performed over the signals affects not the amplitudes themselves but rather the amplitudes with respect to the time axis. We have used discrete-time signals in these examples, but the same applies to continuous-time signals.

Practical Applications

Time-shifting is an important operation that is used in many signal-processing applications. For example, a time-delayed version of the signal is used when performing autocorrelation. (You can learn more about autocorrelation in my previous article, Understanding Correlation.)

Another field that involves the concept of time delay is artificial intelligence, such as in systems that use Time Delay Neural Networks.

2. Time Scaling

Now that we understand more about performing addition and subtraction on the independent variable representing the signal, we'll move on to multiplication.

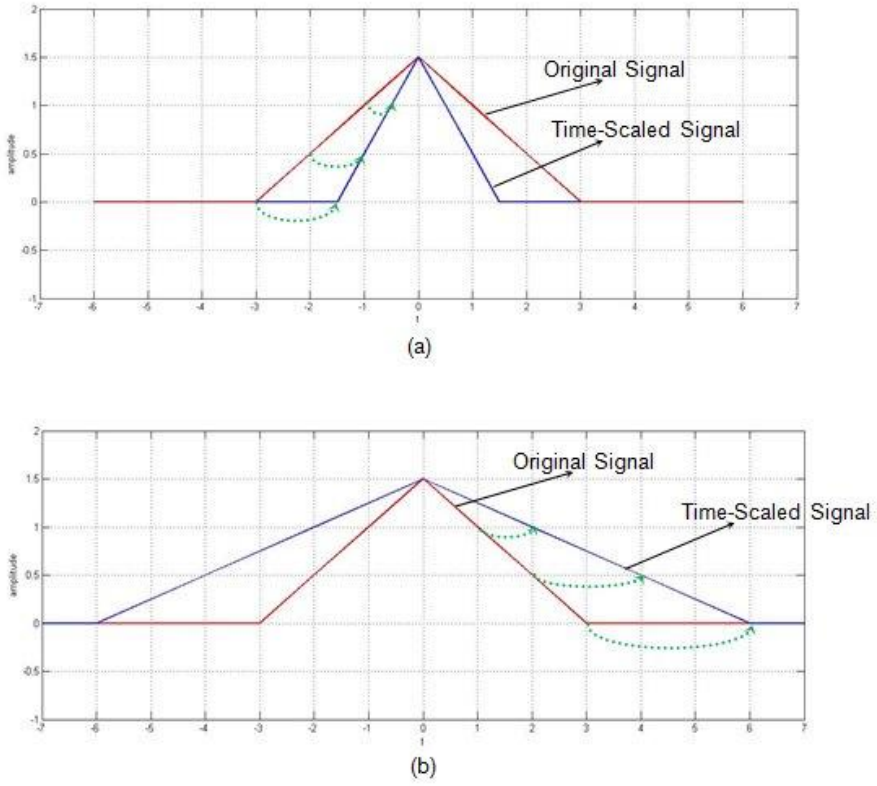
For this, let's consider our input signal to be a continuous-time signal x(t) as shown by the red curve in Figure 3.

Now suppose that we multiply the independent variable (t) by a number greater than one. That is, let's make t in the signal into, say, 2t. The resultant signal will be the one shown by the blue curve in Figure 3.

From the figure, it's clear that the time-scaled signal is contracted with respect to the original one. For example, we can see that the value of the original signal present at t = -3 is present at t = -1.5 and those at t= -2 and at t = -1 are found at t = -1 and at t = -0.5 (shown by green dotted-line curved arrows in the figure).

This means that, if we multiply the time variable by a factor of 2, then we will get our output signal contracted by a factor of 2 along the time axis. Thus, it can be concluded that the multiplication of the signal by a factor of n leads to the compression of the signal by an equivalent factor.

Now, does this mean that dividing the variable t by a number greater than 1 will cause the signal to become expanded? That is, if we divide the variable t by a factor of n, will we get a signal which is stretched by an equivalent factor?



Figure

3. Original signal with its time-scaled versions

Let's check it out.

For this, let's consider our signal to be the same as the one in Figure 3 (the red curve in the figure). Now let's multiply its time-variable t by ½ instead of 2. The resultant signal is shown by the blue curve in Figure 3(b). You can see that, in this time-scaled signal indicated by the green dotted-line arrows in Figure 3(b), we have the values of the original signal present at the time instants t = 1, 2, and 3 to be found at t = 2, 4, and 6.

This means that our time-scaled signal is a stretched-by-a-factor-of-n version of the original signal. So the answer to the question posed above is "yes."

Although we have analyzed the time-scaling operation with respect to a continuous-time signal, this information applies to discrete-time signals as well. However, in the case of discrete-time signals, time-scaling operations are manifested in the form of decimation and interpolation.

Practical Applications

Basically, when we perform time scaling, we change the rate at which the signal is sampled. Changing the sampling rate of a signal is employed in the field of speech processing. A particular example of this would be a time-scaling-algorithm-based system developed to read text to the visually impaired.

Next, the technique of interpolation is used in Geodesic applications (PDF). This is because, in most of these applications, one will be required to find out or predict an unknown parameter from a limited amount of available data.

3. Time Reversal

Until now, we have assumed our independent variable representing the signal to be positive. Why should this be the case? Can't it be negative?

It can be negative. In fact, one can make it negative just by multiplying it by -1. This causes the original signal to flip along its y-axis. That is, it results in the reflection of the signal along its vertical axis of reference. As a result, the operation is aptly known as the time reversal or time reflection of the signal.

For example, let's consider our input signal to be x[n], shown in Figure 4(a). The effect of substituting –n in the place of n results in the signal y[n] as shown in Figure 4(b).

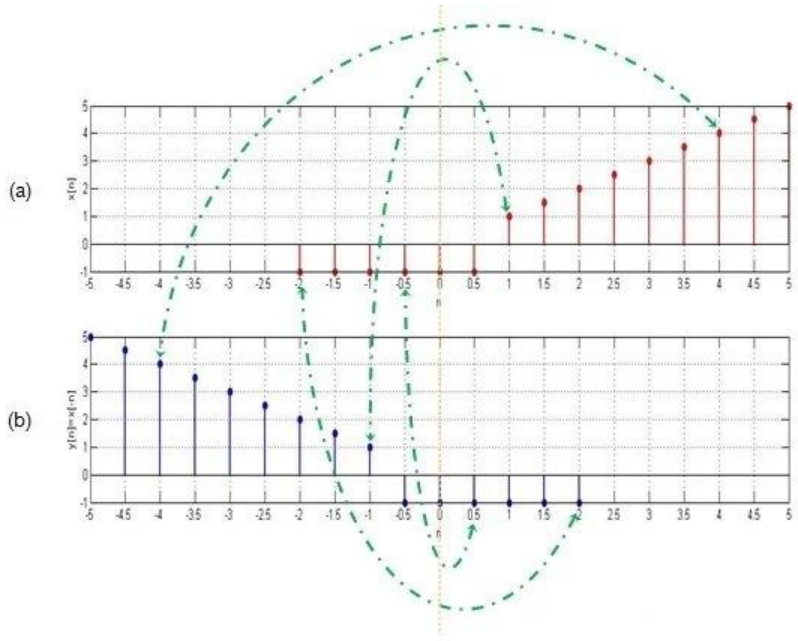


Figure 4. A signal with its reflection

Analog frequency and Digital frequency

The fundamental relation between the analog frequency, Ω , and the digital frequency, ω , is given by the following relation:

$$\omega = \Omega T \qquad (3.3a)$$

or alternately,

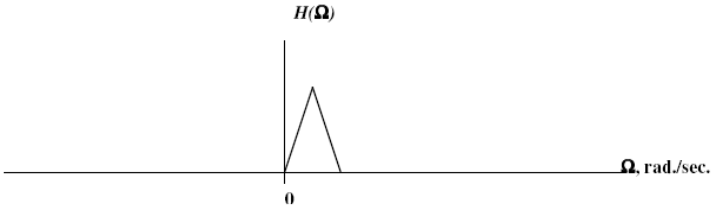
$$\omega = \Omega / f_s \qquad (3.3b)$$

where T is the sampling period, in sec., and fs =1/T is the sampling frequency in Hz.

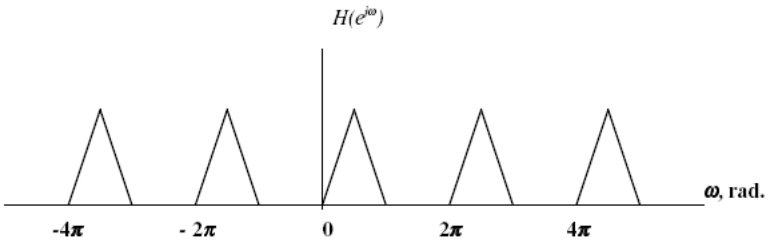
Note, however, the following interesting points:

- The unit of Ω is radian/sec., whereas the unit of ω is just radians.

• The analog frequency, Ω , represents the actual physical frequency of the basic analog signal , for example, an audio signal (0 to 4 kHz) or a video signal (0 to 4 MHz). The digital frequency, ω , is the transformed frequency from Equation 3.3a or Equation 3.3b and can be considered as a mathematical frequency, corresponding to the digital signal.



(a)



(b)

FIGURE 3.1 Analog frequency response and (b) digital frequency response

Definition of Discrete time system

System can be considered as a physical entity which manipulates one or more input signals applied to it. For example a microphone is a system which converts the input acoustic (voice or sound) signal into an electric signal. A system is defined mathematically as a unique operator or transformation that maps an input signal in to an output signal.

This is defined as y(n) = T[x(n)] where x(n) is input signal, y(n) is output signal, T[] is transformation that characterizes the system behavior.

$$y(n) = T\,[x(n)]$$

$$\text{or,} \quad \mathbf{x(n)} \xrightarrow{T} \mathbf{y(n)}$$

Where, T is the general rule or algorithm which is implemented on x(n) or the excitation to get the response y(n). For example, a few systems are represented as,

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

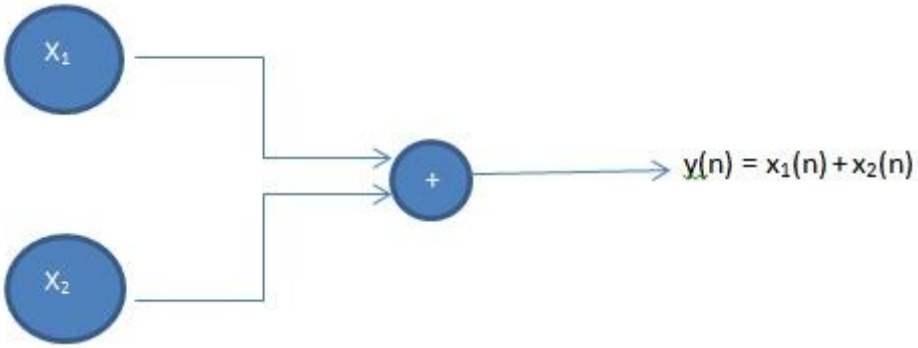
$$\text{or,} \quad y(n) = x(n-1) + x(n) + x(n+1)$$

Block Diagram representation of Discrete-time systems

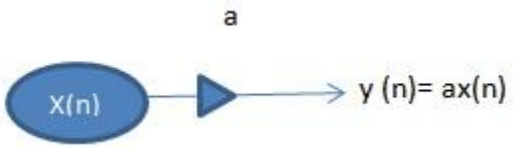
Digital Systems are represented with blocks of different elements or entities connected with arrows which also fulfills the purpose of showing the direction of signal flow,

Some common elements of Discrete-time systems are:-

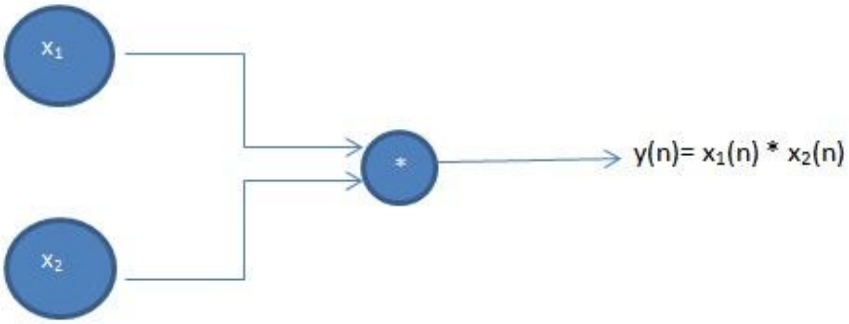
Adder: It performs the addition or summation of two signals or excitation to have a response. An adder is represented as,



Constant Multiplier: This entity multiplies the signal with a constant integer or fraction. And is represented as, in this example the signal x(n) is multiplied with a constant "a" to have the response of the system as y(n).



Signal Multiplier: This element multiplies two signals to obtain one.

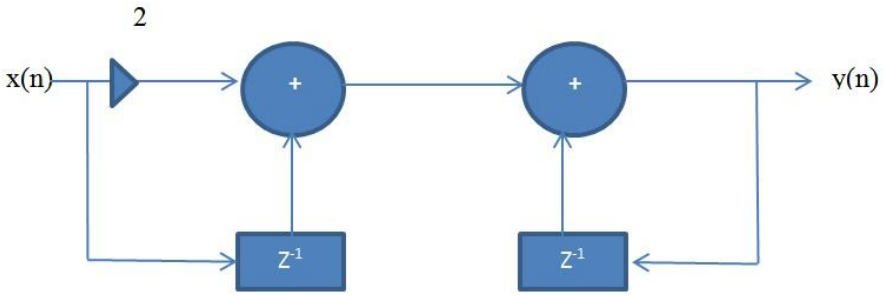


Unit-delay element: This element delays the signal by one sample i.e. the response of the system is the excitation of previous sample. This can element is said to have a memory which stores the excitation at time n-1 and recalls this excitation at the time n form the memory. This element is represented as,

Unit-advance element: This element advances the signal by one sample i.e. the response of the current excitation is the excitation of future sample. Although, as we can see this element is not physically realizable unless the response and the excitation are already in stored or recorded form.

Now that we have understood the basic elements of the Discrete-time systems we can now represent any discrete-time system with the help of block diagram. For example,

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



The above system is an example of Discrete-time system involving the unit delay of current excitation and also one unit delay of the current response of the system.

Classification of Discrete-time Systems

Discrete-time systems are classified on different principles to have a better idea about a particular system, their behavior and ultimately to study the response of the system.

Relaxed system: If $y(n_o -1)$ is the initial condition of a system with response y(n) and $y(n_o -1)=0$, then the system is said to be initially relaxed i.e. if the system has no excitation prior to n_o .

Static and Dynamic systems: A system is said to be a Static discrete-time system if the response of the system depends at most on the current or present excitation and not on the past or future excitation. If there is any other scenario then the system is said to be a Dynamic discrete-time system. The static systems are also said to be memory-less systems and on the other hand dynamic systems have either finite or infinite memory depending on the nature of the system. Examples below will clear any arising doubts regarding static and dynamic systems.

$y(n) = 2x(n) + nx^2(n)$ { static-system }

$y(n) = ax(n)$ { static-system }

$y(n) = ax(n) + bx(n-1) + cx(n+1)$ { dynamic-system with finite memory }

$y(n) = \sum_{k=0}^{n} x(n-k)$ { dynamic-system with finite memory }

$y(n) = \sum_{k=0}^{\infty} x(n-k)$ { dynamic-system with in-finite memory }

The last example is the case of in-finite memory and the others are specified about their type depending on their characteristics.

Time-variant and Time-invariant system: A discrete-time system is said to be time invariant if the input-output characteristics do not change with time, i.e. if the excitation is delayed by k units then the response of the system is also delayed by k units. Let there be a system,

$x(n) \quad \text{----->} \quad y(n) \quad \forall x(n)$

Then the relaxed system T is time-invariant if and only if,

$x(n-k) \quad \text{----->} \quad y(n-k) \quad \forall x(n)$ and k.

Otherwise, the system is said to be time-variant system if it does not follows the above specified set of rules. For example,

$y(n) = ax(n)$ { time-invariant }

$y(n) = x(n) + x(n-3)$ { time-invariant }

$y(n) = nx(n)$ { time-variant }

Note:- In order to check whether the system is time-invariant or time-variant the system must satisfy the "T[x(n-k)]=y(n-k)" condition, i.e. first delay the excitation by k units, then replace n

with (n-k) in the response and then equate L.H.S. and R.H.S. if they are equal then the system is time invariant otherwise not. For example in the last system above,

$$\text{L.H.S.} = T[x(n-k)] = nx(n-k)$$

{not (n-k)x(n-k) which is a general misconfusion}

$$\text{R.H.S.} = y(n-k) = (n-k)\,x(n-k)$$

So, the L.H.S. and R.H.S. are not equal hence the system is time-varient.

Note:- What about Folder, is it a time-variant or time-invariant system, let's see,

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

$$\text{L.H.S.} = y(n-k) = x[-(n-k)] = x(-n+k)$$

$$\text{R.H.S.} = T[x(n-k)] = x(-n-k)$$

Thus, R.H.S. is not equal to L.H.S. so the system is time-variant.

Linear and non-Linear systems: A system is said to be a linear system if it follows the superposition principle i.e. the sum of responses (output) of weighted individual excitations (input) is equal to the response of sum of the weighted excitations. Pay attention to the above specified rule, according to the rule the following condition must be fulfilled by the system in order to be classified as a Linear system,

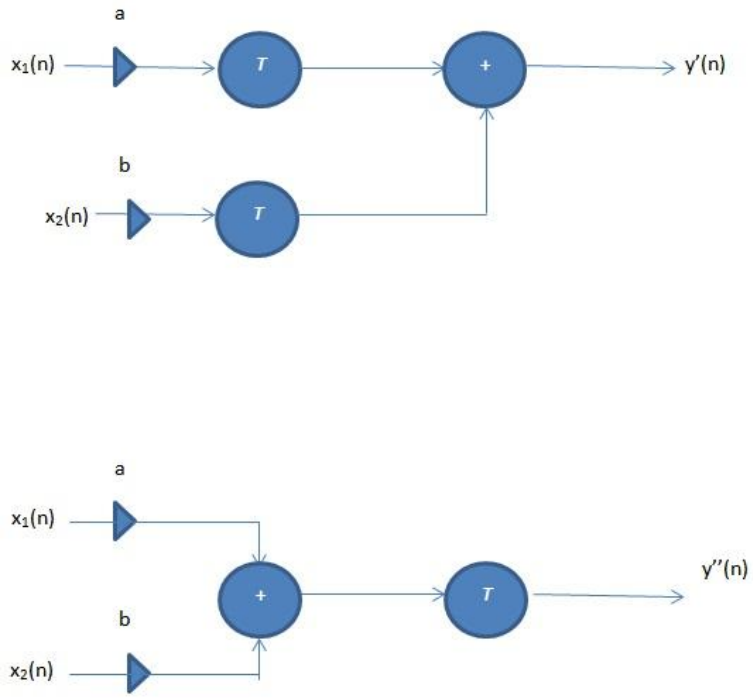
$$\text{If, } y_1(n) = T[\,ax_1(n)\,]$$

$$y_2(n) = T[\,bx_2(n)\,]$$

$$\text{and, } y(n) = T[ax_1(n) + bx_2(n)]$$

Then, the system is said to be linear if ,

$$T[\,ax_1(n) + bx_2(n)] = T[\,ax_1(n)\,] + T[\,bx_2(n)\,]$$



So, if y'(n) = y''(n) then the system is said to be linear. I the system does not fulfills this property then the system is a non-Linear system. For example,

$$y(n) = x(n^2) \qquad \{ \text{linear} \}$$

$$y(n) = Ax(n) + B \qquad \{\text{non – linear}\}$$

$$y(n) = nx(n) \qquad \{ \text{linear} \}$$

The explanation of the above specified examples is left as an exercise for the reader.

Causal and non-Causal systems: A discrete-time system is said to be a causal system if the response or the output of the system at any time depends only on the present or past excitation or input and not on the future inputs. If the system T follows the following relation then the system is said to be causal otherwise it is a non-causal system.

$$y(n) = F\,[x(n), x(n-1), x(n-2), \ldots\ldots]$$

Where F[] is any arbitrary function. A non-causal system has its response dependent on future inputs also which is not physically realizable in a real-time system but can be realized in a recorded system. For example,

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

$y(n) = x(n) + x(n+1)$ {non-Causal }

$y(n) = x(2n)$ { non-Causal since, $y(n) = x(n+n)$ }

Stable and Unstable systems: A system is said to be stable if the bounded input produces a bounded output i.e. the system is BIBO stable. If,

$$x(n) = M \quad \forall \quad -\infty < M < \infty$$

then, $$y(n) = N \quad \forall \quad -\infty < N < \infty$$

Then the system is said to be bounded system and if this is not the case then the system is unbounded or unstable.

ANALYSIS OF DISCRETE-TIME LINEAR TIME-INVARIANT SYSTEMS

Systems are characterized in the time domain simply by their response to a unit sample sequence. Any arbitrary input signal can be decomposed and represented as a weighted sum of unit sample sequences.

Our motivation for the emphasis on the study of LTI systems is twofold. First there is a large collection of mathematical techniques that can be applied to the analysis of LTI systems. Second, many practical systems are either LTI systems or can be approximated by LTI systems.

As a consequence of the linearity and time-invariance properties of the system, the response of the system to any arbitrary input signal can be expressed in terms of the unit sample response of the system. The general form of the expression that relates the unit sample response of the system and the arbitrary input signal to the output signal, called the convolution sum

Thus we are able to determine the output of any linear, time-invariant system to any arbitrary input signal.

There are two basic methods for analyzing the behavior or response of a linear system to a given input signal.

The first method for analyzing the behavior of a linear system to a given input signal is first to decompose or resolve the input signal into a sum of elementary signals. The elementary signals are selected so that the response of the system to each signal component is easily determined. Then, using the linearity property of the system, the responses of the system to the elementary signals are added to obtain the total response of the system to the given input signal.

Suppose that the input signal ***x(n)*** is resolved into a weighted sum of elementary signal components ***{ x_k(n))*** so that

$$x(n) = \sum_k c_k x_k(n)$$

where the ***{c_k}*** is the set of amplitudes (weighting coefficients) in the decomposition of the signal ***x(n) .*** Now suppose that the response of the system to the elementary signal component ***x_k(n)*** is ***y_k(n).*** Thus

$$y_k(n) \equiv \mathcal{T}[x_k(n)]$$

assuming that the system is relaxed and that the response to ***$c_k x_k$(n)*** is ***$c_k v_k$(n)*** as a consequence of the scaling property of the linear system.

Finally, the total response to the input ***x (n)*** is

$$\begin{aligned} y(n) &= \mathcal{T}[x(n)] = \mathcal{T}\left[\sum_k c_k x_k(n)\right] \\ &= \sum c_k \mathcal{T}[x_k(n)] \\ &= \sum_k c_k y_k(n) \end{aligned}$$

In **the above equation** we used the additivity property of the linear system.

Resolution of a Discrete-Time Signal into Impulses

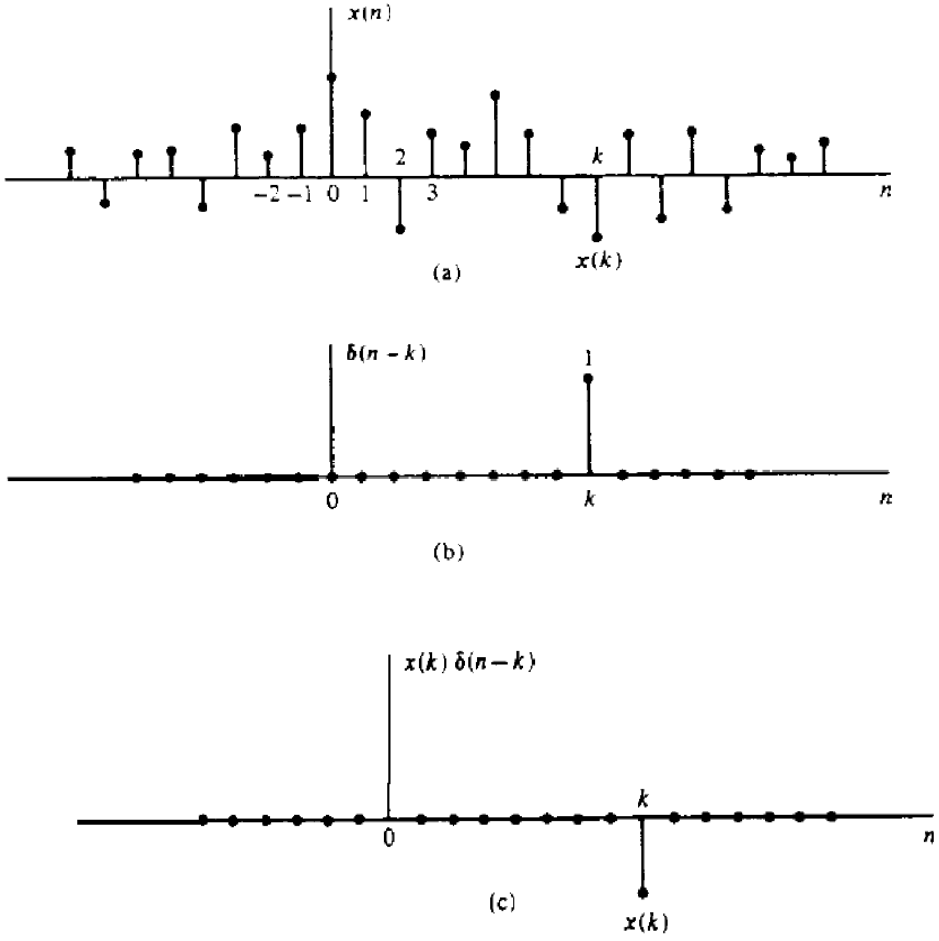
Suppose we have an arbitrary signal ***x(n)*** that we wish to resolve into a sum of unit sample sequences. we

select the elementary signals $x_k(n)$ to be

$$x_k(n) = \delta(n - k)$$

where ***k*** represents the delay of the unit sample sequence. To handle an arbitrary signal ***x(n)*** that may have nonzero values over an infinite duration, the set of unit impulses must also be infinite, to encompass the infinite number of delays.

Now suppose that we multiply the two sequences *x(n)* and *δ(n* - ***k) .*** Since ***δ(n*** - ***k)*** is zero everywhere except at n = ***k .*** where its value is unity, the result of this multiplication is another sequence that is zero everywhere except at ***n*** = ***k.*** where its value is ***x (k) ,*** as illustrated in Fig. below. Thus



Multiplication of a signal *x(n)* with a shifted unit sample sequence.

If we repeat this multiplication over all possible delays, -∞ < k < ∞, and sum all the product sequences, the result will be a sequence equal to the sequence ***x(n) ,*** that is,

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

Example .

Consider the special case of a finite-duration sequence given as

$$x(n) = \{2, 4, 0, 3\}$$
$$\uparrow$$

Resolve the sequence *x (n)* into a sum of weighted impulse sequences.

Solution: Since the sequence *x (n)* is nonzero for the **time instants *n* = -1, 0. 2, we**

need three impulses at delays *k* = - 1. 0, 2. Following (2.3.10) we find that

$$x(n) = 2\delta(n+1) + 4\delta(n) + 3\delta(n-2)$$

Response of LTI Systems to Arbitrary Inputs: The Convolution Sum

we denote the response ***y(n,k)*** of the system to the input unit sample sequence at ***n***= ***k*** by the special symbol ***h(n.*** k), -∞< ***k*** < ∞. That is,

$$y(n, k) \equiv h(n, k) = \mathcal{T}[\delta(n-k)]$$

n is the time index and ***k*** is a parameter showing the location of the input impulse. If the impulse at the input is scaled by an amount ***ck*** ≡**x(*k*)** th***e*** response of the system is the correspondingly scaled output, that is,

$$c_k h(n, k) = x(k)h(n, k)$$

Finally, if the input is the arbitrary signal *x(n)* that is expressed as a sum of

weighted impulses. that is.

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

Then the response of the system to **x(n)** is the corresponding sum of weighted outputs, that is.

$$y(n) = \mathcal{T}[x(n)] = \mathcal{T}\left[\sum_{k=-\infty}^{\infty} x(k)\delta(n-k)\right]$$

$$= \sum_{k=-\infty}^{\infty} x(k)\mathcal{T}[\delta(n-k)]$$

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

The above equation follows from the superposition property of linear systems, and is known as the ***superposition summation.***

In the above equation we used the linearity property of the system bur nor its time invariance property.

Then by the time-invariance property, the response of the system to the delayed unit sample sequence ***δ(n - k)*** is

$$h(n-k) = \mathcal{T}[\delta(n-k)]$$

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

The formula above gives the response *y(n)* of the LTI system as a function of the input signal ***x(n)*** and the unit sample (impulse) response ***h(n)*** is called a **conv*olution sum.***

The process of computing the convolution between ***x(k)*** and ***h(k)*** involves the following four steps.

1. ***Folding.*** Fold ***h(k)*** about ***k*** = 0 to obtain ***h (- k).***

2. ***Shifting.*** Shift ***h (- k)*** by n_o to the right (left) if ***no*** is positive (negative), to obtain $h(n_o - k)$ **.**

3, *Multiplication.* Multiply ***x(k)*** by ***h(no - k)*** to obtain the product sequence

$$v_{no}(k) = x(k)h(n_o - k).$$

4. Summation. Sum all the values of the product sequence $v_{no}(k)$ to obtain the

value of the output at time $n = n_o$.

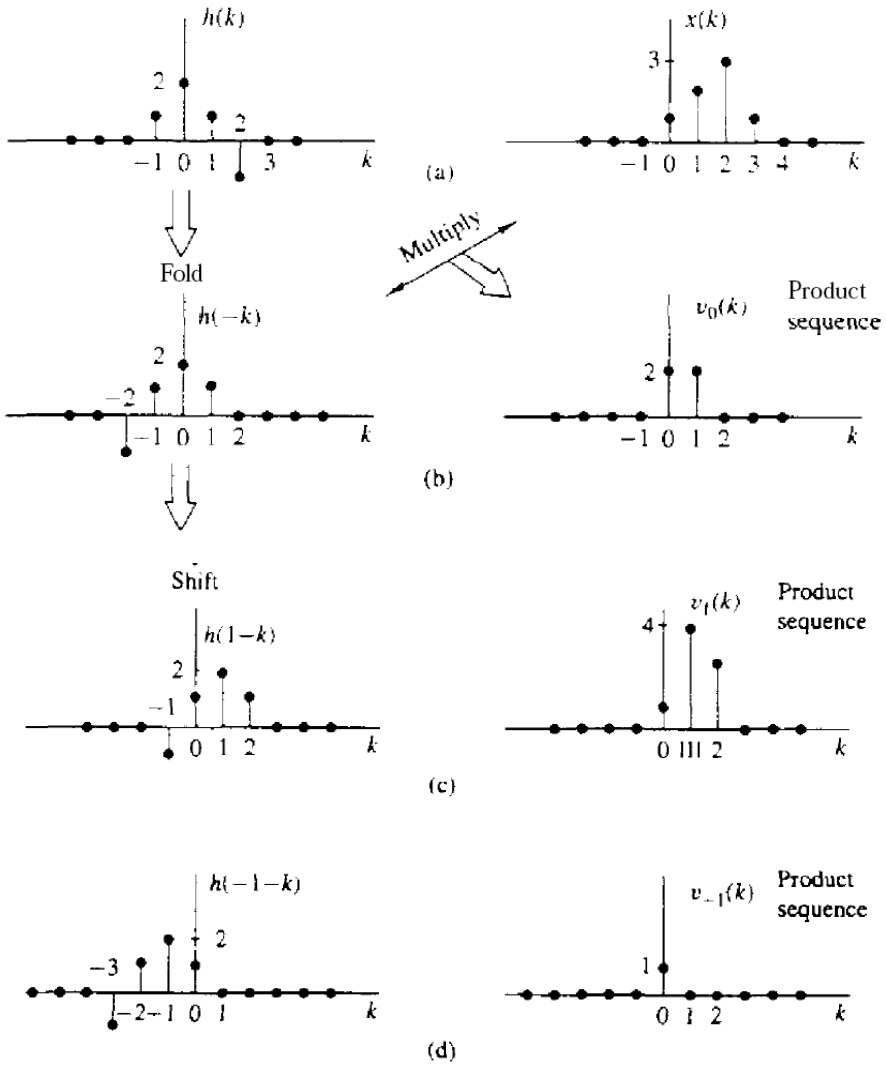
Example .

The impulse response of a linear time-invariant system is

$$h(n) = \{1, 2, 1, -1\}$$
$$\uparrow$$

Determine the response of the system to the input signal

$$x(n) = \{1, 2. 3, 1\}$$
$$\uparrow$$



$$y(1) = \sum_{k=-\infty}^{\infty} v_1(k) = 8$$

$$y(-1) = 1$$

$$y(n) = \{\ldots, 0, 0, \underset{\uparrow}{1}, 4, 8, 8, 3, -2, -1, 0, 0, \ldots\}$$

Filtering using Overlap-save and Overlap-add methods

In many applications one of the signals of a convolution is much longer than the other. For instance when filtering a speech signal $x_L[k]$ of length L with a room impulse response $h_N[k]$ of length $N \ll L$. In order to perform the convolution various techniques have been developed that perform the filtering on limited portions of the signals. These portions are known as partitions, segments or blocks. The respective algorithms are termed as *segmented* or *block-based* algorithms. The following section introduces two techniques for the block-based convolution of signals. The basic concept of these is to divide the convolution $y[k]=x_L[k] * h_N[k]$ into multiple convolutions operating on (overlapping) segments of the signal xL[k].

Overlap-Add Algorithm

The overlap-add algorithm is based on splitting the signal $x_L[k]$ into non-overlapping segments $x_p[k]$ of length P.

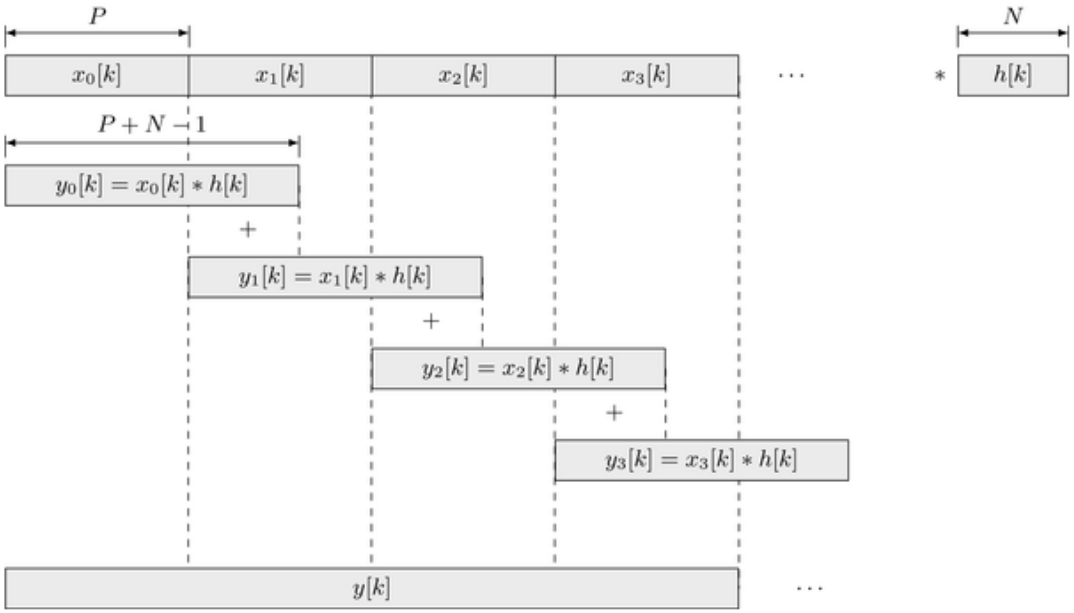
$$x_L[k] = \sum_{p=0}^{L/P-1} x_p[k - p \cdot P]$$

where the segments $x_p[k]$ are defined as

$$x_p[k] = \begin{cases} x_L[k + p \cdot P] & \text{for } k = 0, 1, \ldots, P-1 \\ 0 & \text{otherwise} \end{cases}$$

Note that $x_L[k]$ might have to be zero-padded so that its total length is a multiple of the segment length P. Introducing the segmentation of $x_L[k]$ into the convolution yields where $y_p[k]=x_p[k] * h_N[k]$. This result states that the convolution of $x_L[k]*hN[k]$ can be split into a series of convolutions $y_p[k]$ operating on the samples of one segment(block) only. The length of $y_p[k]$ is N+P−1. The result of the overall convolution is given by summing up the results from

the segments shifted by multiples of the segment length P. This can be interpreted as an overlapped superposition of the results from the segments, as illustrated in the following diagram.



Signal flow of overlap-add algorithm

Overlap-Save Algorithm

The overlap-save algorithm, also known as *overlap-discard algorithm*, follows a different strategy as the overlap-add technique introduced above. It is based on an overlapping segmentation of the input $x_L[k]$ and application of the periodic convolution for the individual segments.

Lets take a closer look at the result of the periodic convolution $x_p[k]*h_N[k]$, where $x_p[k]$ denotes a segment of length P of the input signal and $h_N[k]$ the impulse response of length N. The result of a linear convolution $x_p[k]*h_N[k]$ would be of length P+N−1. The result of the periodic convolution of period P for P>N would suffer from a circular shift (time aliasing) and superposition of the last N−1 samples to the beginning. Hence, the first N−1 samples are not equal to the result of the linear convolution. However, the remaining P−N+1 do so.

This motivates to split the input signal $x_L[k]$ into overlapping segments of length P where the p-th segment overlaps its preceding (p−1)-th segment by N−1 samples.

$$x_p[k] = \begin{cases} x_L[k + p \cdot (P - N + 1) - (N - 1)] & \text{for } k = 0, 1, \ldots, P - 1 \\ 0 & \text{otherwise} \end{cases}$$

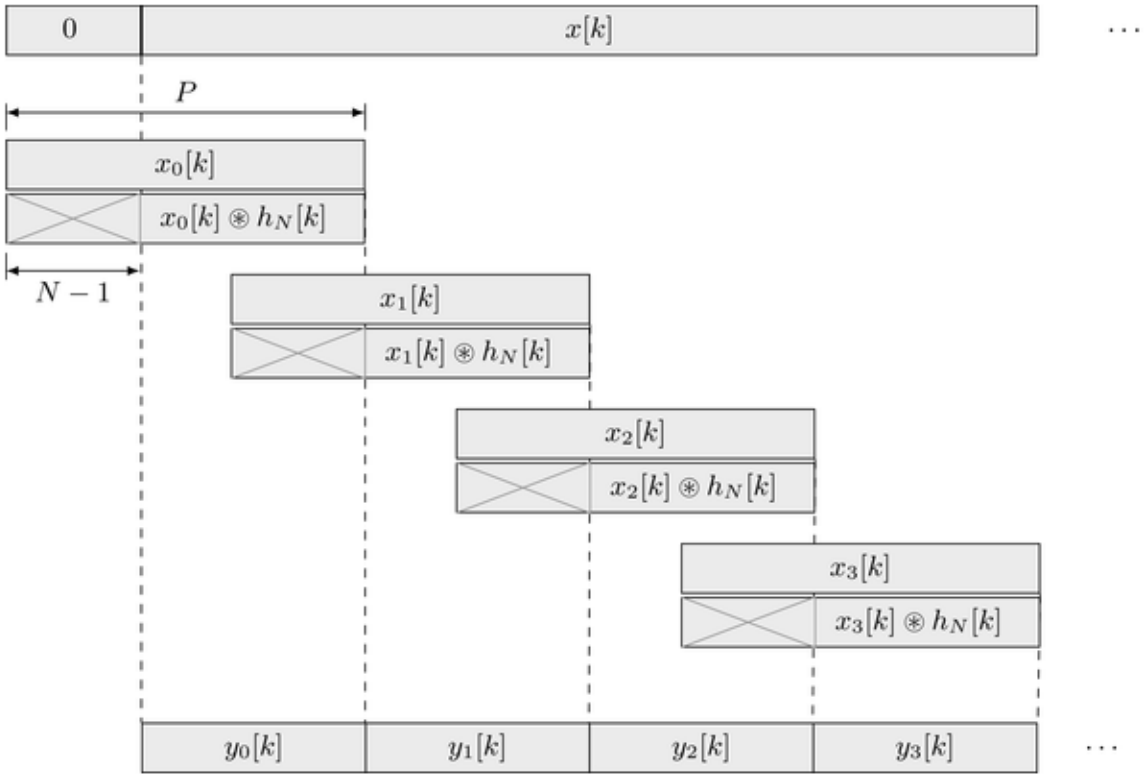
The part of the circular convolution $x_p[k]*h_N[k]$ of one segment $x_p[k]$ with the impulse response $h_N[k]$ that is equal to the linear convolution of both is given as

$$y_p[k] = \begin{cases} x_p[k] \circledast h_N[k] & \text{for } k = N-1, N, \ldots, P-1 \\ 0 & \text{otherwise} \end{cases}$$

The output y[k] is simply the concatenation of the $y_p[k]$

$$y[k] = \sum_{p=0}^{L/P-1} y_p[k - p \cdot (P - N + 1) + (N - 1)]$$

The overlap-save algorithm is illustrated in the following diagram.



Signal flow of overlap-save algorithm

For the first segment $x_0[k]$, N−1 zeros have to be appended to the beginning of the input signal $x_L[k]$ for the overlapped segmentation. From the result of the periodic convolution $x_p[k]*h_N[k]$ the first N−1 samples are discarded, the remaining P−N+1 are copied to the output y[k]. This is indicated by the alternative notation *overlap-discard* used for the technique

Causal Linear Time-Invariant Systems

In the case of a linear time-invariant system, causality can be translated to a condition on the impulse response. To determine this relationship, let us consider a linear time-invariant system having an output at time ***n*** = ***no*** given by the convolution formula

$$y(n_0) = \sum_{k=-\infty}^{\infty} h(k)x(n_0 - k)$$

Suppose that we subdivide the sum into two sets of terms, one set involving present and past values of the input [i.e.. ***x (n)*** for n ≤ n_o] and one set involving future values of the input [i.e., ***x (n) . n*** > n_o]. Thus we obtain

$$\begin{aligned} y(n_0) &= \sum_{k=0}^{\infty} h(k)x(n_0 - k) + \sum_{k=-\infty}^{-1} h(k)x(n_0 - k) \\ &= [h(0)x(n_0) + h(1)x(n_0 - 1) + h(2)x(n_0 - 2) + \cdots] \\ &\quad + [h(-1)x(n_0 + 1) + h(-2)x(n_0 + 2) + \cdots] \end{aligned}$$

We observe that the terms in the first sum involve ***x(n_0), x(n_o - 1) .*** . . . , which are the present and past values of the input signal. On the other hand, the terms in the second sum involve the input signal components ***x(n_o*** + 1), ***x(n_o*** +2). . . . . Now, if the output at time ***n*** = ***no*** is depend only on the present and past inputs, then, clearly. the impulse response of the system must satisfy the condition

$$h(n) = 0 \qquad n < 0$$

Since ***h(n)*** is the response of the relaxed linear time-invariant system to a unit impulse applied at ***n*** = 0, it follows that ***h(n)*** = 0 for n < 0 is both a necessary and a sufficient condition for causality. Hence an *LTI* ***system*** is ***causal if and only if its impulse response is zero for negative values of n.***

Since for a causal system, ***h(n)*** = 0 for ***n*** < 0. the limits on the summation of the convolution formula may be modified to reflect this restriction. Thus we have the two equivalent forms

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

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

Up to this point we have treated linear and time-invariant systems that are characterized by their unit sample response ***h(n).*** In turn h(n) allows us to determine the output ***y(n)*** of the system for any given input sequence ***x(n)*** by means of the convolution summation.

In the case of **FIR systems,** such a realization involves additions, Multiplications, and a finite number of memory locations. Consequently, an FIR system is readily implemented directly, as implied by the convolution summation.

If the system is **IIR**. however, its practical implementation as implied by convolution is clearly impossible. since it requires an infinite number of memory locations, multiplications, and additions. A question that naturally arises, then, is whether or not it is possible to realize IIR systems other than in the form suggested by the convolution summation. Fortunately, the answer is yes.

There is a practical and computationally efficient means for implementing a family of IIR systems, as will be demonstrated in this section, Within the general class of IIR systems. this family of discrete-time systems is more conveniently described by difference equations. This family or subclass of IIR systems is very useful in a variety of practical applications, including the implementation of digital filters, and the modeling of physical phenomena and physical systems.

Recursive and Nonrecursive Discrete-Time Systems

As indicated above, the convolution summation formula expresses the output of the linear time-invariant system explicitly and only in terms of the input signal.

However, this need not be the case, as is shown here. There are many systems where it is either necessary or desirable to express the output of the system not only in terms of the present and past values of the input, but also in terms of the already available past output values. The following problem illustrates this point.

Suppose that we wish to compute the ***cumulative average*** of a signal ***x (n)*** in the interval $0 \leq k \leq n$, defined as

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

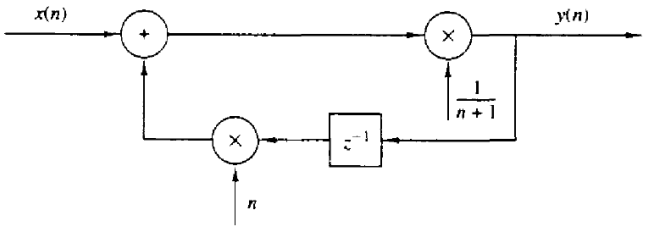
the computation of ***y(n)*** requires the storage of all the input samples ***x (k)*** for $0 \le k \le n$. Since *n* is increasing, our memory requirements grow linearly with time.

Our intuition suggests, however, that *y(n)* can be computed more efficiently

by utilizing the previous output value ***y(n*** - *I)* . Indeed, by a simple algebraic rearrangement , we obtain

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

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



This is an example of a ***recursive system.***

Difference Equations in Discrete Time Systems

Here a treatment of linear difference equations with constant coefficients and it is confined to first- and second-order difference equations and their solution. Higher-order difference equations of this type and their solution is facilitated with the Z–transform

1-Recursive Method for Solving Difference Equations

In mathematics, a *recursion* is an expression, such as a polynomial, each term of which is determined by application of a formula to preceding terms. The solution of a difference equation is often obtained by recursive methods. An example of a recursive method is Newton's method for solving non-linear equations. While recursive methods yield a desired

result, they do not provide a **closed-form** solution. If a closed-form solution is desired, we can solve difference equations using the Method of Undetermined Coefficients, and this method is similar to the classical method of solving linear differential equations with constant coefficients.

2-Method of Undetermined Coefficients

A second-order difference equation has the form

$$y(n) + a_1 y(n-1) + a_2(n-2) = f(n) \qquad (A.1)$$

Where a*1*d a_2 are constants and the right side is some function of *n*. This differenc equation expresses the output *y*(*n*) at time *n* as the linear combination of two previous outputs *y*(*n*-1) and *y*(*n*-2). The right side of relation (A.1) is referred to as the **forcing function** The general (closed-form) solution of relation (A.1) is the same as that used for solving second-order differential equations. The three steps are as follows:

1. Obtain the **natural response** (complementary solution) in terms of two arbitrary real constants k_1 and k_2 , where a_1 and a_2 are also real constants, that is,

$$y_C(n) = k_1 a_1^n + k_2 a_2^n \qquad (A.2)$$

2. Obtain the forced response (particular solution) in terms of an arbitrary real constant $\mathbf{k_3}$ **,** that is,

$$y_P(n) = k_3 a_3^n \qquad (A.3)$$

where the right side of (A.3) is chosen with reference to Table A.1.

TABLE A.1 Forms of the particular solution for different forms of the forcing function

Form of forcing function	Form of particular solution[a]
Constant	k – a constant
an^k – a is a constant	$k_0 + k_1 n + k_2 n^2 + \ldots + k_k n^k$ – k_i is constant
$ab^{\pm n}$ – a and b are constants	Expression proportional to $b^{\pm n}$
$a\cos(n\omega)$ or $a\sin(n\omega)$	$k_1 \cos(n\omega) + k_2 \sin(n\omega)$

3. Add the natural response (complementary solution) $y_c(n)$ and the forced response (particular solution) $y_p(n)$to obtain the total solution, that is,

$$y(n) = y_C(n) + y_P(n) = k_1 a_1^n + k_2 a_2^n + y_P(n) \qquad (A.4)$$

4. Solve for k_1 and k_2 in (A.4) using the given initial conditions. It is important to remember that the constants k_1 and k_2 must be evaluated from the total solution of (A.4), not from the complementary solution $y_c(n)$**.**

Example 1

Find the total solution for the second−order difference equation

$$y(n) - \frac{5}{6}y(n-1) + \frac{1}{6}y(n-2) = 5^{-n} \qquad n \geq 0 \qquad (A.5)$$

subject to the initial conditions $y(-2) = 25$ and $y(-1) = 6$.

Solution:

1. We assume that the complementary solution $y_c(n)$ has the form

$$y_C(n) = k_1 a_1^n + k_2 a_2^n \qquad (A.6)$$

The homogeneous equation of (A.5) is

$$y(n) - \frac{5}{6}y(n-1) + \frac{1}{6}y(n-2) = 0 \qquad n \geq 0 \qquad (A.7)$$

Substitution of

$$y(n) = a^n$$

into (A.7) yields

$$a^n - \frac{5}{6}a^{n-1} + \frac{1}{6}a^{n-2} = 0 \qquad (A.8)$$

Division of (A.8) by

a^{n-2}

Yields

$$a^2 - \frac{5}{6}a + \frac{1}{6} = 0 \qquad \text{(A.9)}$$

The roots of (A.9) are

$$a_1 = \frac{1}{2} \qquad a_2 = \frac{1}{3} \qquad \text{(A.10)}$$

and by substitution into (A.6) we obtain

$$y_C(n) = k_1\left(\frac{1}{2}\right)^n + k_2\left(\frac{1}{3}\right)^n = k_1 2^{-n} + k_2 3^{-n} \qquad \text{(A.11)}$$

2. Since the forcing function is

5^{-n},

, we assume that the particular solution is

$$y_p(n) = k_3 5^{-n} \qquad \text{(A.12)}$$

and by substitution into (A.5),

$$k_3 5^{-n} - k_3\left(\frac{5}{6}\right)5^{-(n-1)} + k_3\left(\frac{1}{6}\right)5^{-(n-2)} = 5^{-n}$$

Division of both sides by

5^{-n}

Yields

$$k_3\left[1 - \left(\frac{5}{6}\right)5 + \left(\frac{1}{6}\right)5^2\right] = 1$$

Or k=1 and thus

$$y_p(n) = 5^{-n} \qquad \text{(A.13)}$$

The total solution is the addition of (A.11) and (A.13), that is,

$$y(n) = y_C(n) + y_p(n) = k_1 2^{-n} + k_2 3^{-n} + 5^{-n} \qquad \text{(A.14)}$$

To evaluate the constants k_1 and k_2 we use the given initial conditions, i.e., s $y(-2) = 25$ and $y(-1) = 6$. For $n = -2$, (A.14) reduces to

$$y(-2) = k_1 2^2 + k_2 3^2 + 5^2 = 25$$

from which

$$4k_1 + 9k_2 = 0 \qquad \text{(A.15)}$$

For $n = -1$, (A.14) reduces to

$$y(-1) = k_1 2^1 + k_2 3^1 + 5^1 = 6$$

from which

$$2k_1 + 3k_2 = 1 \qquad \text{(A.16)}$$

Simultaneous solution of (A.15) and (A.16) yields

$$k_1 = \frac{3}{2} \qquad k_2 = -\frac{2}{3} \qquad \text{(A.17)}$$

and by substitution into (A.14) we obtain the total solution as

$$y(n) = y_C(n) + y_p(n) = \left(\frac{3}{2}\right)2^{-n} + \left(-\frac{2}{3}\right)3^{-n} + 5^{-n} \qquad n \geq 0 \qquad \text{(A.18)}$$

IMPLEMENTATION OF DISCRETE-TIME SYSTEMS

In practice, system design and implementation are usuaHy treated jointly rather than separately. Often, the system design is driven by the method of implementation and by implementation constraints, such as cost. hardware limitations, size limitations, and power requirements. At this point, we have not as yet developed the necessary analysis and design tools to treat such complex issues. However, we have developed sufficient background to consider some basic implementation methods for realizations of LTI systems described by linear constant-coefficient difference equations.

Structures for the Realization of Linear Time-Invariant Systems

In this subsection we describe structures for the realization of systems described by linear constant-coefficient difference equations.

As a beginning, let us consider the first-order system

$$y(n) = -a_1 y(n-1) + b_0 x(n) + b_1 x(n-1)$$

which is realized as in Fig. a. This realization uses separate delays (memory) for both the input and output signal samples and it is called a ***direct form*** I ***structure.***

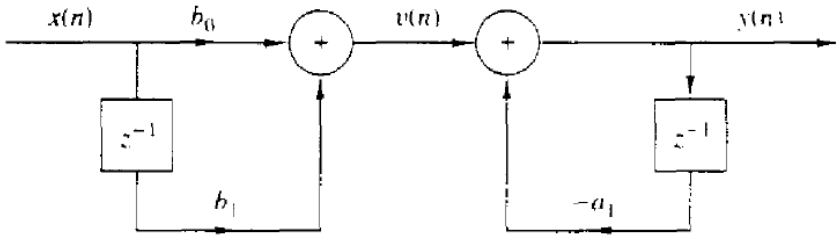
Note that this system can be viewed as two linear time-invariant systems in cascade.

The first is a nonrecursive, system described by the equation

$$v(n) = b_0 x(n) + b_1 x(n-1)$$

whereas the second is a recursive system described by the equation

$$y(n) = -a_1 y(n-1) + v(n)$$



(a)

Thus if we interchange the order of the recursive and nonrecursive systems, we obtain an alternative structure for the realization of the system described above**.** The resulting system is shown in Fig. b. From this figyre we obtain

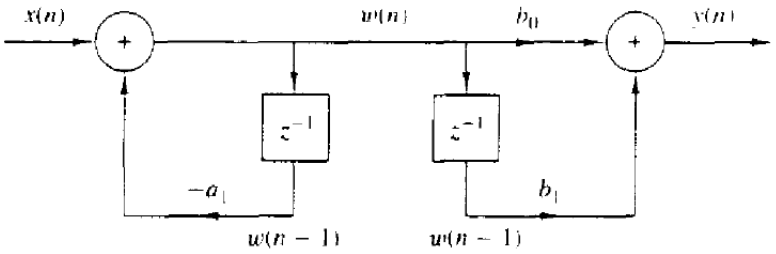
the two difference equations

$$w(n) = -a_1 w(n-1) + x(n)$$
$$y(n) = b_0 w(n) + b_1 w(n-1)$$

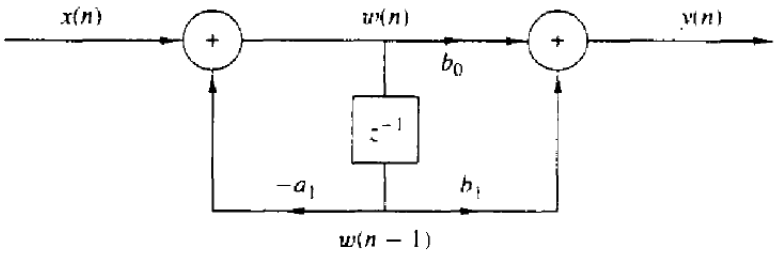
which provide an alternative algorithm for computing the output of the system described by the single difference equation given first. In other words. The last two difference equations are equivalent to the single difference equation .

A close observation of Fig. **a,b** reveals that the two delay elements contain the same input ***w(n)*** and hence the same output ***w(n-1)***. Consequently. these

two elements can be merged into one delay, as shown in Fig. **c.** In contrast



(b)



(c)

to the direct form I structure, this new realization requires only one delay for the auxiliary quantity ***w(n),*** and hence it is more efficient in terms of memory requirements. It is called the ***direct*** form 11 structure and it is used extensively in practical applications. These structures can readily be generalized for the general linear time-invariant recursive system described by the difference equation

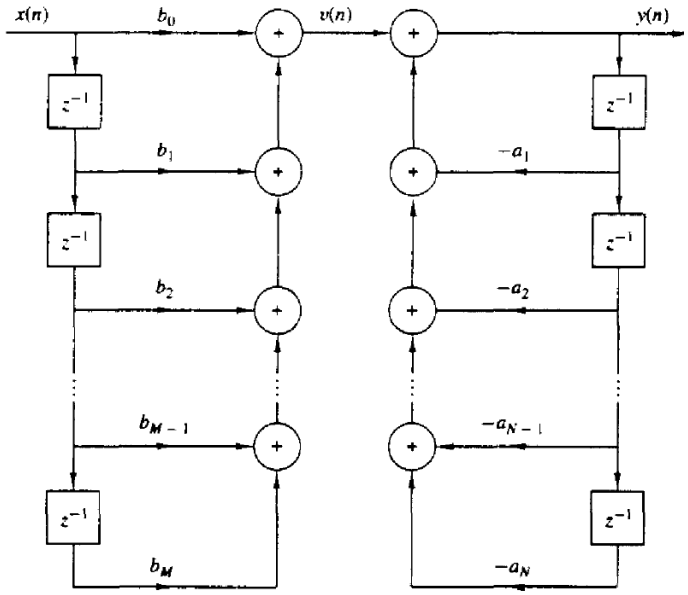
$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k x(n-k)$$

Figure below illustrates the direct form I structure for this system. This structure requires M + N delays and N + ***M*** + 1 multiplications. It can be viewed as the cascade of a nonrecursive system

$$v(n) = \sum_{k=0}^{M} b_k x(n-k)$$

and a recursive system

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + v(n)$$



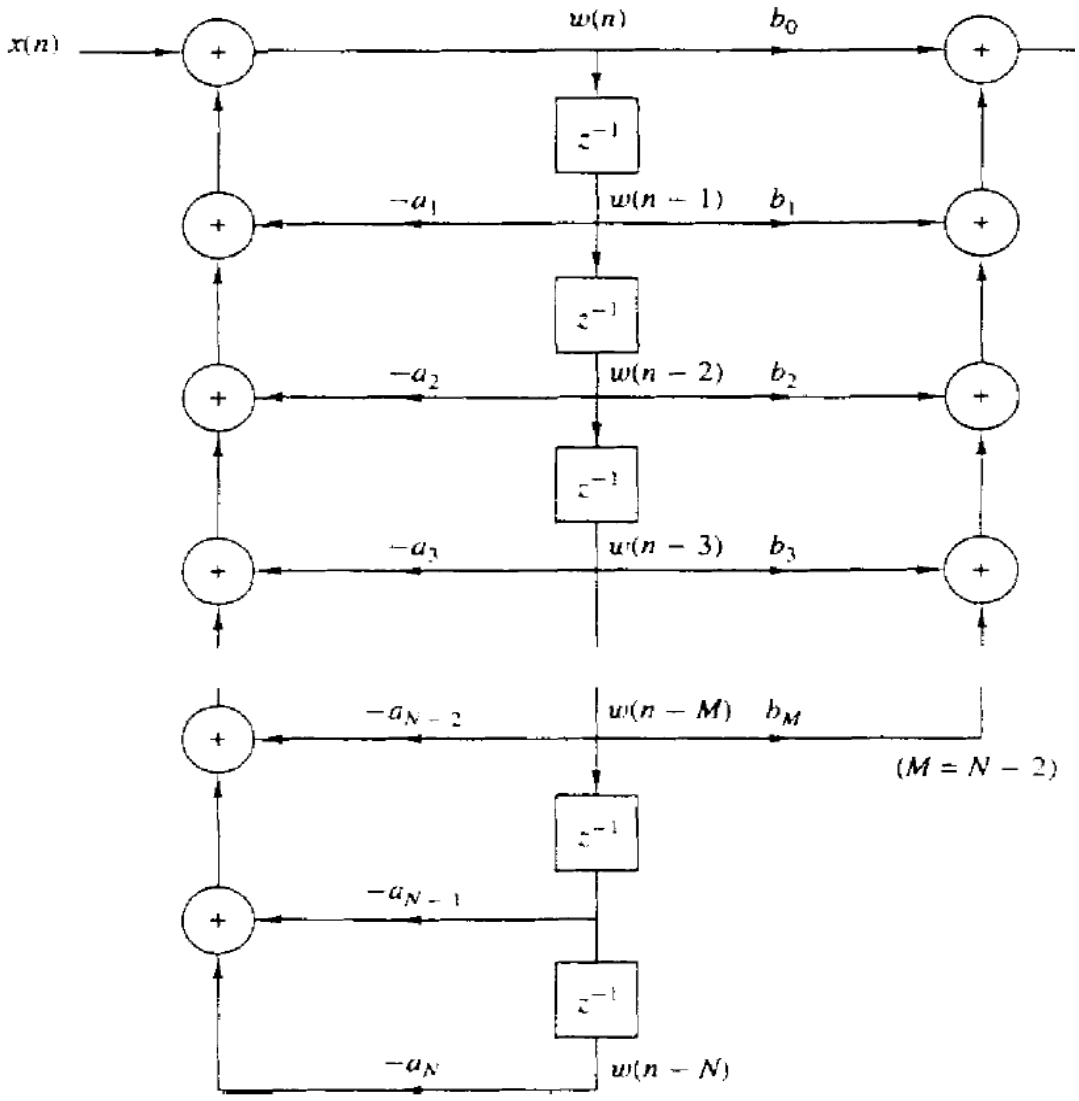
By reversing the order of these two systems as was previously done for the first-order system, **we** obtain the direct form **I1** structure shown in Fig. below for N > M. This structure is the cascade of a recursive system

$$w(n) = -\sum_{k=1}^{N} a_k w(n-k) + x(n)$$

followed **by** a nonrecursive system

$$y(n) = \sum_{k=0}^{M} b_k w(n-k)$$

We observe that if N $\geq$ M, this structure requires a number of delays equal to the order N of the system. However, if M > ***N,*** the required memory is specified by M. Figure above can easily by modified to handle this case. Thus the direct form **I1** structure requires M + ***N*** + 1 multiplications and max(M, N} delays. Because it requires the minimum number of delays for the realization of the system described by *given difference equation.*


x(n)
w(n)
b0
z⁻¹
−a1
w(n − 1)
b1
z⁻¹
−a2
w(n − 2)
b2
z⁻¹
−a3
w(n − 3)
b3
−aN − 2
w(n − M)
bM
(M = N − 2)
z⁻¹
−aN − 1
z⁻¹
−aN
w(n − N)

UNIT-II

DISCRETE FOURIER TRANSFORM AND EFFICIENT COMPUTATIONS

CONTINUOUS TIME FOURIER SERIES (CTFS)

x(t) is the Continuous Time Periodic signal with period is T sec , x(t) can be expressed as weighted sum of complex exponential signals.

$$x(t) = \sum_{k=-\infty}^{\infty} c_k \, e^{j\Omega_0 kt} \quad \text{for all k values} \qquad (1)$$

$$c_k = \frac{1}{T}\int_0^T x(t)\, e^{-j\Omega_0 kt}\, dt \quad \text{for all k}$$

where Ω_0 is fundamental frequency of signal f(t) (2)

Since frequency range of continuous time signals extended from $-\infty \; to \; \infty$ x(t) consisting of infinite number of frequency components , where spacing between successive harmonically related frequencies is $\frac{1}{T}$ and where T is fundamental time period .

DISCRETE TIME FOURIER SERIES (DFS)

Similar way discrete time period signal x(n) with period N can be expressed as weighted sum of complex exponential sequences. The range of frequencies for x(n) is unique over the interval (0 , 2π) or (– π , + π). Consequently Fourier series representation of x(n) will contain N (finite) frequency components and spacing between two successive frequency components is $\frac{2\pi}{N}$ radians. Therefore Fourier series representation of periodic sequence need only contain N of these complex exponentials.

$$x(n) = \sum_{k=0}^{N-1} c_k \, e^{j\frac{2\pi}{N}kn} \quad k = 0,1,\ldots\ldots..N\text{-}1 \qquad (3)$$

where $\{c_k\}$ are Fourier coefficients

It contains N harmonically related exponential functions.

Derivation for c_k

Multiplying the equation 1 on both sides by $e^{-j\frac{2\pi}{N}ln}$

$$x(n)\, e^{-j\frac{2\pi}{N}ln} = \sum_{k=0}^{N-1} c_k \, e^{j\frac{2\pi}{N}(k-l)n} \quad k = 0,1,\ldots\ldots..N\text{-}1$$

Summing the products on both sides from n = 0 to n = N-1

$$\sum_{n=0}^{N-1} \mathrm{x(n)}\, e^{-j\frac{2\pi}{N}ln} = \sum_{n=0}^{N-1}\sum_{k=0}^{N-1} c_k\, e^{j\frac{2\pi}{N}(k-l)n} \qquad (4)$$

Interchange the order of summation

$$\sum_{n=0}^{N-1} \mathrm{x(n)}\, e^{-j\frac{2\pi}{N}ln} = \sum_{k=0}^{N-1} c_k \sum_{n=0}^{N\_1} e^{j\frac{2\pi}{N}(k-l)n}$$

$$\sum_{n=0}^{N-1} e^{j\frac{2\pi}{N}(k-l)n} = \begin{cases} N\ for\ \ k-l = rN\ \ r\ integer \\ 0\ for\ otherwise \end{cases} \qquad (5)$$

$$\mathrm{N}\, c_l = \sum_{n=0}^{N-1} \mathrm{x(n)}\, e^{-j\frac{2\pi}{N}ln} \quad l = 0,1,2,\ldots\ldots N-1 \qquad (6)$$

$$c_k = X(k) = \frac{1}{N} \sum_{n=0}^{N-1} \mathrm{x(n)} e^{-j\frac{2\pi}{N}kn} \quad k = 0,1,2,\ldots\ldots N-1 \quad analysis\ eqation \quad (7)$$

$$x(n) = \sum_{n=0}^{N-1} \mathrm{X(k)} e^{-j\frac{2\pi}{N}kn} \quad n = 0,1,2,\ldots\ldots N-1 \quad sythesis\ \ eqation \qquad 8$$

$$W_N = e^{-j\frac{2\pi}{N}}$$

$X(k)$Represents amplitude and phase associate with frequency component

$$X(k+N) = X(k)$$

Fourier coefficients form a periodic sequence when extended outside of the range k = 0, 1, 2, 3…..N-1.Thus spectrum of the periodic signal also periodic sequence with period N. In frequency domain to covering the fundamental range $0 \leq \omega_k = \frac{2\pi}{N}k \leq 2\pi$ for $k = 0,1,2,\ldots\ldots N-1$ .in contrast, the frequency range $-\pi \leq \omega_k \leq \pi$ corresponds to $-\frac{N}{2} \leq k \leq \frac{N}{2}$ which creates incontinence when N is odd. Clearly if we use sampling frequency F_s *the* range $\mathrm{k} = 0,1,2,3\ldots..\mathrm{N}-1$ corresponds to the frequency range 0 ≤ F <F_s.

Power Density Spectrum of Periodic Signal

Average power of discrete time signals with period N is defined as

$$P_{av} = \frac{1}{N}\sum_{n=0}^{N-1} x(n)x^*(n) \qquad 9$$

$$= \frac{1}{N}\sum_{n=0}^{N-1} x(n) \sum_{k=0}^{N-1} \mathrm{X}^*(\mathrm{k}) e^{j\frac{2\pi}{N}kn}$$

Interchange the order of two summations

$$P_{av} = \sum_{k=0}^{N-1} \mathrm{X}^*(\mathrm{k}) \ \frac{1}{N} \sum_{n=0}^{N-1} x(n) e^{j\frac{2\pi}{N}kn} \qquad 10$$

$$= \sum_{k=0}^{N-1} \mathrm{X}^*(\mathrm{k}) \ X(k)$$

$$= \sum_{k=0}^{N-1} |\mathrm{X}^2(\mathrm{k})|$$

It shows that average power in the signal is the sum of the powers of the individual frequency components. The sequence $\mathrm{X}^2(\mathrm{k})$ is the distribution of power as function of frequency and is called as power density spectrum of the period signal.

Energy of the sequence x(n) over one period given by

$$\sum_{n=0}^{N-1} |\mathrm{x}^2(\mathrm{n})| = N \sum_{k=0}^{N-1} |\mathrm{X}^2(\mathrm{k})| \qquad 11$$

PROPERTIES OF DFS

Linearity

Let us consider two periodic sequences $\widetilde{x_1(n)} and \widetilde{x_2(n)}$, both with period equal to N

$$\widetilde{x_1(n)} \overset{DFS}{\longleftrightarrow} \widetilde{X_1(k)}$$

$$\widetilde{x_2(n)} \overset{DFS}{\longleftrightarrow} \widetilde{X_2(k)}$$

$$\widetilde{x_3(n)} \overset{DFS}{\longleftrightarrow} \widetilde{X_3(k)}$$

$$\widetilde{x_3(n)} = a\,\widetilde{x_1(n)} + b\widetilde{x_2(n)}$$

$\widetilde{x_3(n)}$ *is defind as linear combination* $\widetilde{x_1(n)} and \widetilde{x_2(n)}$ With period equal to N

$$\widetilde{X_3(k)} = \sum_{n=0}^{N-1} \widetilde{x_3(n)}\, {W_N}^{nk} for\ k = 0,1,2,3 \ldots\ldots\ldots\ldots\ldots N-1$$

$$= \sum_{n=0}^{N-1} (a\,\widetilde{x_1(n)} + b\widetilde{x_2(n)})\, {W_N}^{nk}$$

$$= \sum_{n=0}^{N-1} a\,\widetilde{x_1(n)}\ {W_N}^{nk} + \sum_{n=0}^{N-1} b\widetilde{x_2(n)})\, {W_N}^{nk}$$

$$= a \sum_{n=0}^{N-1} \widetilde{x_1(n)}\ {W_N}^{nk} + b \sum_{n=0}^{N-1} \widetilde{x_2(n)}\ {W_N}^{nk}$$

$$\widetilde{X_3(k)} = a\,\widetilde{X_1(k)} + b\widetilde{X_2(k)}$$

DFS of linear combination of two sequences is equal to linear combination of DFS of individual sequences and all sequences are periodic with period N.

Shifting of sequence

x(n) is the periodic sequence with period N

$$\widetilde{x(n)} \overset{DFS}{\longleftrightarrow} \widetilde{X(k)}$$

$\widetilde{x_1(n)} = \widetilde{x(n-m)}$ *is the shifted version of* $x(n)$ *by amount* m *to right*

$$\widetilde{X_1(k)} = \sum_{n=0}^{N-1} x(\widetilde{n-m})\, W_N{}^{nk} \; for\; k = 0,1,2,3 \ldots\ldots\ldots\ldots\ldots N-1$$

$$n - m = l\,, l + m = n$$

$$\widetilde{X_1(k)} = \sum_{l=-m}^{N-1-m} \widetilde{x(l)}\, W_N{}^{(l+m)k} for\; k = 0,1,2,3 \ldots\ldots\ldots\ldots\ldots N-1$$

Fourier coefficients of periodic sequence are a periodic sequence so

$$\widetilde{X_1(k)} = W_N{}^{mk} \textstyle\sum_{l=0}^{N-1} \widetilde{x(l)}\, W_N{}^{lk} \qquad for\; k = 0,1,2,3 \ldots\ldots\ldots\ldots\ldots N-1$$

$$\widetilde{X_1(k)} = W_N{}^{mk} \widetilde{X(k)}$$

$$\widetilde{x(n-m)} \overset{DFS}{\longleftrightarrow} W_N{}^{mk} \widetilde{X(k)}$$

Similar results applies shift in Fourier coefficients of sequence $W_N{}^{mk}\widetilde{x(n)}$

$$W_N{}^{mk} \widetilde{x(n)} = \widetilde{X(k-l)}$$

$$W_N{}^{mk} \widetilde{x(n)} \overset{DFS}{\longleftrightarrow} \widetilde{X(k-l)}$$

Any shift greater than the period cannot distinguish in time domain from shorter shift.

Shift of x(n) defined on 0 to N-1 by amount of m to right denoted by x((n-m) modulo N).This operation , wrapping part fall outside of region of interest around front of sequence or just straight translation of period extension outside of 0 to N-1 of given sequence.

$$x((n-m))_N = x((n-m) \qquad modulo\; N) = [x(N-m), x(N-m+1), \ldots\ldots\ldots\ldots x(N-m-1)]$$

If m=N

$$x((n-N))_N = x((n-N) modulo\ N) = [x(0), x(1), \ldots\ldots\ldots\ldots x(N-1)]$$

Which is same as original sequence x (n)

Periodic convolution

$let\ \widetilde{x_1(n)} and\ \widetilde{x_2(n)}$ be the two periodic sequences of period N and its DFS are $\widetilde{X_1(k)}\ and\ \widetilde{X_2(k)}$ respectively.

$$\widetilde{X_3(k)} = \widetilde{X_1(k)} \,.\, \widetilde{X_2(k)}$$

$$= \sum_{m=0}^{N-1} \widetilde{x_1(m)}\ {W_N}^{mk} \sum_{r=0}^{N-1} \widetilde{x_2(r)} {W_N}^{rk}$$

$$= \sum_{m=0}^{N-1} \sum_{r=0}^{N-1} \widetilde{x_1(m)}\ \widetilde{x_2(r)} {W_N}^{(r+m)k}$$

$$\widetilde{x_3(n)} = \frac{1}{N} \sum_{k=0}^{N-1} \widetilde{X_3(k)}\ {W_N}^{-nk}\ for\ n = 01,2,3 \ldots\ldots\ldots\ldots N-1$$

$$\widetilde{x_3(n)} = \frac{1}{N} \sum_{k=0}^{N-1} \sum_{m=0}^{N-1} \sum_{r=0}^{N-1} \widetilde{x_1(m)}\ \widetilde{x_2(r)} {W_N}^{(r+m-n)k} \quad for\ n = 01,2,3 \ldots\ldots\ldots\ldots N-1$$

$$\widetilde{x_3(n)} = \frac{1}{N} \sum_{k=0}^{N-1} \sum_{m=0}^{N-1} \sum_{r=0}^{N-1} \widetilde{x_1(m)}\ \widetilde{x_2(r)} {W_N}^{-(n-m-r)k} \quad for\ n = 01,2,3 \ldots\ldots\ldots\ldots N-1$$

$$\sum_{k=0}^{N-1} {W_N}^{-(n-m-r)k} = \begin{cases} N & for\ n-m-r = lN \\ o & for\ otherwise \end{cases}$$

n-m = r + lN, $x(n-m) = x(r+lN) = x(r)$ because of periodic property

$$\widetilde{x_3(n)} = \sum_{m=0}^{N-1} \widetilde{x_1(m)}\ \widetilde{x_2(n-m)} \quad for\ n = 01,2,3 \ldots\ldots\ldots\ldots N-1$$

In above equation $\widetilde{x_1(m)} and\ \widetilde{x_2(n-m)}$ are periodic in m with period N and consequently their product. Also summation carried out only **over a one period**. This type convolution commonly referred periodic convolution.

Product in time domain

$let\ \widetilde{x_1(n)} and\ \widetilde{x_2(n)}$ be the two periodic sequences of period N and its DFS are $\widetilde{X_1(k)}\ and\ \widetilde{X_2(k)}$ respectively.

$$\widetilde{x_3(n)} = \widetilde{x_1(n)} \,.\, \widetilde{x_2(n)}$$

$$= \frac{1}{N}\sum_{m=0}^{N-1} \widetilde{X_1(m)}\ W_N^{-mn}\ \ \frac{1}{N}\sum_{r=0}^{N-1} \widetilde{X_2(r)} W_N^{-rn}$$

DFs of $\widetilde{x_3(n)}$ is $\widetilde{X_3(k)} = \sum_{n=0}^{N-1} \widetilde{x_3(n)}\ W_N^{nk}$

$$\widetilde{X_3(k)} = \frac{1}{N}\sum_{n=0}^{N-1}\sum_{m=0}^{N-1}\sum_{r=0}^{N-1}\frac{1}{N}\widetilde{X(m)}\ \widetilde{X_2(r)} W_N^{(k-m-r)n} \quad for\ k = 01,2,3 \ldots\ldots\ldots\ldots . N-1$$

$$\sum_{n=0}^{N-1} W_N^{(k-m-r)n} = \begin{cases} N & for\ k-m-r = lN \\ o & for\ otherwise \end{cases}$$

$$k - m = r + lN, x(k-m) = \widetilde{x(r+lN)} = \widetilde{x(r)}$$

$$\widetilde{X_3(k)} = \frac{1}{N}\sum_{m=0}^{N-1} \widetilde{X_1(m)}\ X_2\widetilde{(k-m)} \quad for\ k = 01,2,3 \ldots\ldots\ldots\ldots . N-1$$

Discrete Time Fourier Transform (DTFT)

Fourier Transform of an aperiodic Discrete Time sequence x(n) represented by function of complex exponential sequence X ($e^{j\omega}$) where $\boldsymbol{\omega}$ is a frequency variable.

DTFT of x(n) is $X(e^{j\omega}) = \sum_{n=-\infty}^{+\infty} x(n)\, e^{-j\omega n}$

DTFT maps time domain sequence in continuous function of sequence

RHS equation shows Fourier series representation of period function X ($e^{j\omega}$) therefore

$$x(n) = \frac{1}{2\pi}\int_{-\pi}^{+\pi} X(e^{j\omega})\, e^{j\omega n} d\boldsymbol{\omega} \quad \boldsymbol{Inverse\ DTFT}$$

Finite energy signal have continuous spectrum

Frequency domain sampling

Let $x(n)$ be the non-periodic discrete time sequence and its Fourier transform $X(\omega)$ is periodic function of $\boldsymbol{\omega}$ with period is 2π radians.

$$X(\omega) = \sum_{n=-\infty}^{+\infty} x(n)\, e^{-j\omega n}$$

Sample the periodic function X ($\boldsymbol{\omega}$) and spacing between two successive samples is $\delta\boldsymbol{\omega} = \frac{2\pi}{N}$ where N is number of samples in interval $0 \leq \omega \leq 2\pi$

K th sample at $\omega_k = \frac{2\pi}{N} k$ where k = 0,1,2,3 ,……..N-1

$X(\frac{2\pi}{N} k) = \sum_{n=-\infty}^{+\infty} x(n)\, e^{-j\omega n}$ where k = 0,1,2, ……..N-1

Summation can be divided into infinite number of summations and each summation contain N terms only

$$X(\frac{2\pi}{N} k) = \cdots\ldots + \sum_{n=-N}^{-1} x(n)\, e^{-j\omega n} + \sum_{n=0}^{N-1} x(n)\, e^{-j\omega n} + \sum_{n=N}^{2N-1} x(n)\, e^{-j\omega n} + \ldots\ldots\ldots\ldots + \sum_{n=lN}^{lN+N-1} x(n)\, e^{-j\omega n}$$

$$= \sum_{l=-\infty}^{+\infty} \sum_{n=lN}^{lN+N-1} x(n)\, e^{-j\omega n}$$

Change Index of inner summation from n to $n - lN$

$$= \sum_{l=-\infty}^{+\infty} \sum_{n=0}^{N-1} x(n - lN)\, e^{-j\omega(n-lN)}$$

Change order of summation

$$= \sum_{n=0}^{N-1} \sum_{l=-\infty}^{+\infty} x(n - lN)\, e^{-j\omega(n-lN)}$$

$$= \sum_{n=0}^{N-1} x_p(n)\, e^{-j\omega(n-lN)}$$

$$x_p(n) = \sum_{l=-\infty}^{+\infty} x(n - lN)$$

$$X(\frac{2\pi}{N} k) = \sum_{n=0}^{N-1} x_p(n)\, e^{-j\frac{2\pi}{N}k(n-lN)}$$

$$X(\frac{2\pi}{N} k) = \sum_{n=0}^{N-1} x_p(n)\, e^{-j\frac{2\pi}{N}kn}$$

$x_p(n)$ *is periodic enxtension of* $x(n)$ *every N samples with fundamental period is N*

$x_p(n) = \sum_{n=0}^{N-1} c_k\, e^{j\frac{2\pi}{N}kn}$ Where c_k *is fourier coecieffients*

$$c_k = \frac{1}{N}\sum_{n=0}^{N-1} x_p(n)\, e^{-j\frac{2\pi}{N}kn}$$

Comparing

$$X(\frac{2\pi}{N}k\ ) = Nc_k$$

$$x_p(n) = \frac{1}{N}\sum_{n=0}^{N-1} \mathrm{X}(\frac{2\pi}{N}k\ )\, e^{j\frac{2\pi}{N}kn}$$

$$\mathrm{X}(\frac{2\pi}{N}k\ ) = \sum_{n=0}^{N-1} x_p(n)\, e^{-j\frac{2\pi}{N}k(n-lN)}$$

This indicates reconstruction of $x_p(n)$ from samples of spectrum X (***ω***) but it does not implies that we can recover x(n) from the samples.

x(n) is obtained from $x_p(n)$ if there is no aliasing , one period of periodic signal $x_p(n)$ is x(n)

x(n) = $x_p(n)$ for $0 \le n \le N-1$

= 0 other wise

So that x(n) is recovered from $x_p(n)$ without ambiguity

If period of periodic sequence $x_p(n)$ is less than length of x(n) , it is not possible recover x(n) from $x_p(n)$ due to time domain aliasing .

Interpolation formula

Spectrum of X(***ω***) in terms of samples $\mathrm{X}(\frac{2\pi}{N}k\ )$ where k = 0,1,2,----------N-1

$$x(n) = \frac{1}{N}\sum_{k=0}^{N-1} \mathrm{X}(\frac{2\pi}{N}k\ )\, e^{j\frac{2\pi}{N}kn}$$

$$X(\omega) = \sum_{n=0}^{N-1} \frac{1}{N} \sum_{k=0}^{N-1} \mathrm{X}(\frac{2\pi}{N}k\ ) e^{j\frac{2\pi}{N}kn}\ \ e^{-j\omega n}$$

$$X(\omega) = \sum_{k=0}^{N-1} \frac{1}{N} \sum_{n=0}^{N-1} \mathrm{X}(\frac{2\pi}{N}k\ )\ \ e^{-j(\omega-\frac{2\pi k}{N})n}$$

$$P(\omega) = \frac{1}{N}\sum_{n=0}^{N-1} e^{-j\omega n} = \frac{1}{N}\frac{1-e^{-j\omega N}}{1-e^{-j\omega}} = \frac{\sin \omega N/2}{N \sin \omega/2} e^{-j\omega(\frac{N-1}{2})}$$

$$P\left(\frac{2\pi}{N}k\right) = \begin{cases} 1 & for \quad k = 0 \\ 0 & for\ k = 1,2 \ldots N-1 \end{cases}$$

Interpolation shifted by $\frac{2\pi}{N}k$ in frequency

$$X(\omega) = \sum_{k=0}^{N-1} \mathrm{X}(\frac{2\pi}{N}k\ )\ \ P\left(\omega - \frac{2\pi}{N}k\right)\ \ for\ N \geq L$$

Phase shift reflects the signal x(n) is a causal , finite duration sequence of length N

Interpolation formula in above expression gives samples values $\mathrm{X}\left(\frac{2\pi}{N}k\ \right) for\ \omega = \frac{2\pi}{N}k. at\ other\ frequenccies$ weighted sum of the original spectral samples.

The Discrete Fourier Transform (DFT)

With frequency domain sampling of aperiodic finite energy sequence, the equally spaced samples $\mathrm{X}\left(\frac{2\pi}{N}k\right)$ for k = 0,1,2,3,……N-1 , do not uniquely represent original sequence x(n) when x(n) has infinite duration instead , the frequency samples $\mathrm{X}\left(\frac{2\pi}{N}k\right)$ correspond to periodic sequence $x_p(n)$ of period N, where $x_p(n)$ periodic extension of x(n)

$$x_p(n) = \sum_{n=-\infty}^{n=\infty} x(n - lN)$$

When x(n) is finite duration of length L ≤ N , then $x_p(n)$ is periodic repetition of x(n)

$$x_p(n) = \begin{cases} x(n) & for \quad 0 \le n \le L-1 \\ 0 & for\ L \le n \le N-1 \end{cases}$$

X(n) = $x_p(n)$ *over a one period only ,the origional sequence* $x(n)$ *can be obtained* from

$frequency\ samples$.

By adding N-L zeros to original sequence, it becomes N point sequence this operation known as zero padding and it does not provide additional information.

Fourier transform of finite length sequence x(n) given by

$X(\omega) = \sum_{n=0}^{N-1} x(n)e^{-j\omega n} \quad where\ 0 \le \omega \le 2\pi$

Sample the $X(\omega)$ at equally spaced frequencies samples

$\omega_k = \frac{2\pi}{N}k\ where\ k = 0,1,2,3, \dots\dots..N-1 \qquad N \ge L\ \ then$

$X\left(\frac{2\pi}{N}k\right) = X(k) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}kn} \quad where\ k = 0,1,2,3, \dots\dots..N-1$ Since x (n) = 0 for

N ≥L this is called **DFT** of sequence x(n)

To recover the x(n) from frequency samples X(k)

$$x(n) = \frac{1}{N}\sum_{n=0}^{N-1} X(k)e^{j\frac{2\pi}{N}kn} \quad where\ n = 0,1,2,3, \dots\dots..N-1 \quad this\ is\ called\ as\ \boldsymbol{inverse}\ DFT$$

$Twiddle\ factor = W_N = e^{-j\frac{2\pi}{N}}$

DFT: $X(k) = \sum_{n=0}^{N-1} x(n)\ {W_N}^{nk} \qquad where\ k = 0,1,2,3, \dots\dots..N-1$

IDFT: $x(n) = \frac{1}{N}\sum_{n=0}^{N-1} X(k){W_N}^{-nk} \qquad where\ n = 0,1,2,3, \dots\dots..\ N-1$

Relationship to the Z transform

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

ROC includes in unit circle, if $X(z)$ sampled N equally spaced samples on the unit circle

$z_k = e^{\frac{j2\pi k}{N}}$ where k = 0,1,2 ……N-1

$$X(z)|_{z=e^{\frac{j2\pi k}{N}}} = \sum_{n=-\infty}^{\infty} x(n) e^{\frac{-j2\pi kn}{N}}$$

This is identical to fourier transform $X(\omega)$ evaluated at the N equally spaced frequencies

$\omega_k = \frac{2\pi}{N} k$ where k =0,1,2, …….N-1

Relationship between $X(z)$ and $X(k)$

$$X(z) = \sum_{n=0}^{N-1} \frac{1}{N} \sum_{k=0}^{N-1} X(k) e^{j\frac{2\pi}{N}kn} \; z^{-n}$$

$$X(z) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) \sum_{n=0}^{N-1} (e^{j\frac{2\pi}{N}k} z^{-1})^n$$

$$X(z) = \frac{1 - z^{-N}}{N} \sum_{k=0}^{N-1} \frac{X(k)}{1 - e^{j\frac{2\pi}{N}k} z^{-1}}$$

$$X(e^{j\omega}) = \frac{1-e^{j\omega N}}{N} \sum_{k=0}^{N-1} \frac{X(k)}{1-e^{j(\omega - \frac{2\pi}{N}k)}}$$

Fourier transform of finite duration sequence

Properties of DFT

Periodicity:

$$x(n) \xleftrightarrow{N-point\ DFT} X(k)$$

$$x(n) = x(n+N) \;\; for\ all\ n\ values$$

$$X(k) = X(k+N) \;\; for\ aal\ k\ values$$

Above condition satisfies the periodicity

According to definition of DFT

$$X(k+N) = \sum_{n=0}^{N-1} x(n) W_N{}^{n(k+N)} \qquad where\ k = 0,1,2,3, \ldots \ldots N-1$$

$$= \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}nk}\ e^{-j2\pi n}$$

$$= \sum_{n=0}^{N-1} x(n) W_N{}^{nk} \qquad \text{Because } e^{-j2\pi n} = 1$$

$$X(k+N) = X(k)$$

$$similarly\ x(n) = x(n+N)$$

Linearity

Let us consider two N point sequences say

$x_1(n) and\ x_2(n)\ are\ N\ point\ sequences\ whose$ DFTs $X_1(k) and\ X_2(k)$ respectively

$$X_1(k) = \sum_{n=0}^{N-1} x_1(n) W_N{}^{nk} \qquad where\ k = 0,1,2,3, \ldots \ldots N-1$$

$$X_2(k) = \sum_{n=0}^{N-1} x_1(n) W_N{}^{nk} \qquad where\ k = 0,1,2,3, \ldots \ldots N-1$$

$$x_3(n) = \text{a}\ x_1(n) + b\ x_2(n)$$

$$X_3(k) = \sum_{n=0}^{N-1} x_3(n) W_N{}^{nk} \qquad where\ k = 0,1,2,3, \ldots \ldots N-1$$

$$X_3(k) = \sum_{n=0}^{N-1} [\text{a}\ x_1(n) + b\ x_2(n)] W_N{}^{nk} \qquad where\ k = 0,1,2,3, \ldots \ldots N-1$$

$$X_3(k) = aX_1(k) + b\ X_2(k)$$

$$\text{a}\ x_1(n) + b\ x_2(n) \overset{DFT}{\longleftrightarrow} \text{a}\ X_1(k) + b\ X_2(k)$$

Where a and b are any real or complex valued constants.

Note: If duration of $x_1(n) and\ x_2(n)$ is not equal say N_1 and N_2 respectively Then duration of linear combination of two sequences is N = max (N_1 , N_2).

Circular symmetry:

Let us Consider finite duration sequence $x(n)$ of length $L \leq N$

$x_p(n)$ *is a* Periodic extension of $x(n)$ with period N

$x_p(n) = \sum_{n=-\infty}^{n=\infty} x(n - lN)$ *where l any interger number*

Shifted version of periodic sequence is ${x_p}^1(n) = x_p(n-k)$

Shifted version finite duration sequence $x_1(n) = x(n-k)$ = one period of periodic sequence ${x_p}^1(n)$ in the range of $0 \leq n \leq N-1$.

$$x_1(n) = \begin{cases} {x_p}^1(n) \; for \;\; 0 \leq n \leq N-1 \\ 0 \quad for\ other\ wise \end{cases}$$

$x_1(n)$ Does not corresponds to linear shift of original sequence $x(n)$ *in* fact both sequences are confined to the interval of $0 \leq n \leq N-1$.it observed that shifting of periodic sequence and examining the interval $0 \leq n \leq N-1$ as a sample leaves the interval at one end it enters the other end. Thus circular shift of an N point sequence is equivalent to a linear shift of its periodic extension and vice versa. Circular shift of $x(n)$ can be represented by index modulo N.

$$x((n-k))_N = x((n-k) mod\ N)$$

Counter clock wise direction has been selected as positive and clock wise direction selected as negative.

Circularly even symmetric N point sequence about the point zero on the circle.

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

Circularly odd symmetric N point sequence about the point zero on the circle

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

Time reversal: an N point sequence is attained by reversing its samples about the point zero on the circle. Time reversal is equivalent to plotting the $x(n)$ in clock wise direction on the circle.

$$x((-n))_N = x(N-n) \; for \;\; 0 \leq n \leq N-1$$

For periodic sequences

Even : $x_p(n) = x_p(-n) = x_p(N-n)$

Odd: $x_p(n) = x_p(-n) = -x_p(N-n)$

If periodic sequence is complex valued

Conjugate even: $x_p(n) = x_p^*(N-n)$

Conjugate odd: $x_p(n) = -x_p^*(N-n)$

Decomposition of $x_p(n)$

$$x_p(n) = x_{pe}(n) + x_{po}(n)$$

$$x_{pe}(n) = \frac{1}{2}[x_p(n) + x_p^*(N-n)$$

$$x_{po}(n) = \frac{1}{2}[x_p(n) - x_p^*(N-n)]$$

Symmetry properties

Let us assume that N-point sequence x(n) and its DFT X(k) are both complex valued then the sequences are expressed as follows

$$x(n) = x_R(n) + j\, x_I(n) \qquad 0 \le n \le N$$

$$X(k) = X_R(k) + j\, X_I(k) \qquad 0 \le k \le N$$

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j\frac{2\pi}{N}kn} \quad where\ k = 0,1,2,3, \ldots \ldots N-1$$

$$X_R(k) + j\, X_I(k) = \sum_{n=0}^{N-1} [x_R(n) + j\, x_I(n)][\cos\frac{2\pi}{N}nk - j\, \sin\frac{2\pi}{N}nk]$$

Comparing real and imaginary terms on both sides

$$X_R(k) = \sum_{n=0}^{N-1} [x_R(n)\cos(\frac{2\pi}{N}nk) + x_I(n)\sin(\frac{2\pi}{N}nk)$$

$$X_I(k) = -\sum_{n=0}^{N-1} [x_R(n)\sin(\frac{2\pi}{N}nk) - x_I(n)\cos(\frac{2\pi}{N}nk)$$

$$x(n) = x_R(n) + j\, x_I(n) \quad = \frac{1}{N}\sum_{n=0}^{N-1}(X_R(k) + j\, X_I(k))[\quad \cos\frac{2\pi}{N}nk + j\ \sin\frac{2\pi}{N}nk]$$

$$x_R(n) = \frac{1}{N}\ \sum_{n=0}^{N-1}\ [X_R(k)\ \cos\frac{2\pi}{N}nk - X_I(k)\ \sin\frac{2\pi}{N}nk]$$

$$x_I(n) = \frac{1}{N}\sum_{n=0}^{N-1}\ [X_R(k)\ \sin(\frac{2\pi}{N}nk) + X_I(k)\ \cos(\frac{2\pi}{N}nk)]$$

Real valued sequences

If x(n) is real valued sequence

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}kn} \quad where\ k = 0,1,2,3, \ldots \ldots N-1$$

$$X(N-k) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}(N-k)n}$$

$$= \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}(-k)n}$$

$$X(N-k) = X(-k)$$

$$X(k) = \sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}kn} \quad where\ k = 0,1,2,3, \ldots \ldots N-1$$

$$(X(k))^* = \left(\sum_{n=0}^{N-1} x(n)e^{-j\frac{2\pi}{N}kn} \quad \right)^*$$

$$= \sum_{n=0}^{N-1} x(n)e^{j\frac{2\pi}{N}kn}$$

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

If x (n) is real and even

$$x(n) = \ x(N-n) \quad 0 \leq n \ \leq N$$

x(n) is real and even then $x_I(n) = 0$ and $X_I(k) = 0$

$$X(k) = \sum_{n=0}^{N-1} x(n) \cos(\frac{2\pi}{N} nk) \quad where\ k = 1,2,3, \ldots\ldots\ldots\ldots N-1$$

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) \cos(\frac{2\pi}{N} nk) \quad where\ n = 1,2,3, \ldots\ldots\ldots\ldots N-1$$

if x(n) is real and odd

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

$$X_R(k) = 0$$

$$X(k) = -j \sum_{n=0}^{N-1} x(n) \sin(\frac{2\pi}{N} nk) \quad where\ k = 1,2,3, \ldots\ldots\ldots\ldots N-1$$

$$x(n) = j\frac{1}{N} \sum_{k=0}^{N-1} X(k) \sin(\frac{2\pi}{N} nk) \quad where\ n = 1,2,3, \ldots\ldots\ldots\ldots N-1$$

Pure imaginary sequences

$$x(n) = jx_I(n)$$

$$X_R(k) = \sum_{n=0}^{N-1} x_I(n) \sin(\frac{2\pi}{N} nk) \text{ odd}$$

$$X_I(k) = \sum_{n=0}^{N-1} x_I(n) \cos(\frac{2\pi}{N} nk) \text{ even}$$

$$x_I(n)\ is\ odd\ X_I(k) = 0\ X(k) purely\ real\ and\ x_I(n)\ is\ even\ X_R(k) = 0\ and\ hence\ X(k) purely imaginary$$

Multiplication of two DFTs and circular convolution

Let us consider two finite duration sequences of length N say $x_1(n) and\ x_2(n), and\ their\ N-point\ DFT$ are $X_1(k) and\ X_2(k)$ respectively, if we multiply two DFTs result is a DFT say $X_3(k)$of a sequence $x_3(n)\ of\ length$ N.

$$X_3(k) = X_1(k)\ X_2(k)$$

$$IDFT\ OF\ X_3(k)\ is\ x_3(m) = \frac{1}{N} \sum_{k=0}^{N-1} X_1(k)\ X_2(k)\ e^{j\frac{2\pi}{N} km}$$

$$= \frac{1}{N}\sum_{k=0}^{N-1}\left[\sum_{n=0}^{N-1} x_1(n)\ e^{-j\frac{2\pi}{N}kn}\right]\left[\sum_{l=0}^{N-1} x_2(l)\ e^{-j\frac{2\pi}{N}kl}\right]\ e^{j\frac{2\pi}{N}km}$$

$$= \frac{1}{N}\sum_{n=0}^{N-1}\left[\sum_{l=0}^{N-1} x_1(n)\ x_2(l)\ \right]\left[\sum_{k=0}^{N-1} e^{j\frac{2\pi}{N}k(m-n-l)}\ \right]$$

$$\sum_{k=0}^{N-1} e^{j\frac{2\pi}{N}k(m-n-l)} = \begin{cases} N \text{ for } m-n-l = rN \text{ where } r \text{ is integer} \\ 0 \qquad\qquad\qquad\qquad \text{for otherwise} \end{cases}$$

$$x_3\ (m) = \sum_{n=0}^{N-1} x_1(n)\ x_2((m-n))_N$$

Multiplication of two DFT sequences results DFT of circular convolution of two time domain sequences.

Note: if length of two sequences are not equal in such case N = Max of two sequences and apply zero padding to smaller sequence.

Time reversal property

Let us consider N point sequence x (n) and it time reversal $\mathrm{x}((-\mathrm{n}))_N = x(N-n)$

$$DFT\ of\ x(N-n) = \sum_{n=0}^{N-1} \mathrm{x}(\mathrm{N}-\mathrm{n})\ e^{-j\frac{2\pi}{N}kn}$$

Index N-n changed to m

$$= \sum_{n=0}^{N-1} \mathrm{x}(\mathrm{m})\ e^{-j\frac{2\pi}{N}k(N-m)}$$

$$= \sum_{n=0}^{N-1} \mathrm{x}(\mathrm{m})\ e^{j\frac{2\pi}{N}km}$$

$$= \sum_{n=0}^{N-1} \mathrm{x}(\mathrm{m})\ e^{-j\frac{2\pi}{N}(k-N)m}$$

From definition of DFT

$$DFT\ of\ x(N-n) = X(N-k) = \mathrm{X}((-\mathrm{k}))_N \text{ for } 0 \le \mathrm{k} \le \mathrm{N}\text{-}1$$

Hence reversing N point sequence in time is equivalent to reversing the DFT values.

Circular shift operation:

X(k) is N point DFT of sequence x(n) of duration N

$$DFT\ of\ x((n-l))_N = \sum_{n=0}^{N-1} \mathrm{x}((\mathrm{n}-\mathrm{l}))_N\ e^{-j\frac{2\pi}{N}kn}$$

$$= \sum_{n=0}^{l-1} x((n-l))_N \, e^{-j\frac{2\pi}{N}kn} + \sum_{n=l}^{N-1} x(n-l) \quad e^{-j\frac{2\pi}{N}kn}$$

But $x\big((n-l)\big)_N = x(N-l+n)$

$$\sum_{n=0}^{l-1} x((n-l))_N \, e^{-j\frac{2\pi}{N}kn} = \sum_{n=0}^{l-1} x(N+n-l) \quad e^{-j\frac{2\pi}{N}kn}$$ if m = N+n-l

$$\sum_{n=0}^{l-1} x((n-l))_N \, e^{-j\frac{2\pi}{N}kn} = \sum_{m=N-l}^{N-1} x(m) \quad e^{-j\frac{2\pi}{N}k(m+l)}$$

$$\sum_{n=l}^{N-1} x(n-l) \quad e^{-j\frac{2\pi}{N}kn}$$

If n-l = m then $\sum_{m=0}^{N-1-l} x(m) \quad e^{-j\frac{2\pi}{N}k(m+l)}$

$$DFT\ of\ x\big((n-l)\big) = \sum_{m=0}^{N-1-l} x(m) \quad e^{-j\frac{2\pi}{N}k(m+l)} + \sum_{m=N-l}^{N-1} x(m) \quad e^{-j\frac{2\pi}{N}k(m+l)}$$

$$= = \sum_{m=0}^{N-1} x(m) \quad e^{-j\frac{2\pi}{N}k(m+l)} = = e^{-j\frac{2\pi}{N}kl} \quad X(k)$$

Circular shifting in time domain results, multiplication of DFT of sequence x(n) with $e^{-j\frac{2\pi}{N}kl}$ similarly this is dual to Circular shifting in frequency domain results, multiplication of sequence x(n) with $e^{-j\frac{2\pi}{N}kl}$.

Circular correlation

X(k) and Y(k) are N point DFTs of x(n) and y(n) respectively

DFT of correlation of two sequences x(n) and y(n) is given by $r_{xy}(l)$

$$r_{xy}(l) = \sum_{n=0}^{N-1} x(n)\ y^*((n-l)_N$$

$r_{xy}(l) = x(n)\ N\ y*(-l)$ similar to circular convolution

$$DFT\ of\ r_{xy}(l) = X(k)Y^*(k)$$

Multiplication of two sequences

$x_1(n)$ *and* $x_2(n)$ *are two N point sequences and its DFTs* $X_1(k)$ *and* $X_2(k)$ *respectively*
DFT of $x_1(n).x_2(n) = \frac{1}{N} X_1(k)\ N\ X_2(k)$

Parseval's theorem

$$r_{xy}(l) = \sum_{n=0}^{N-1} x(n)\ y^*((n-l)_N$$

If l=0

$$r_{xy}(0) = \sum_{n=0}^{N-1} x(n)\ y^*((n)_N$$

$$r_{xy}(\mathrm{l}) = \frac{1}{N} \sum_{k=0}^{N-1} \quad R_{xy}(k)\ e^{j\frac{2\pi}{N}kl}$$

$$r_{xy}(\mathrm{l}) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)\ y*(k) \quad e^{j\frac{2\pi}{N}kl}$$

$$r_{xy}(0) = r_{xy}(\mathrm{l}) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)\ y*(k)$$

Special case of parseaval's x(n) = y(n)

$$r_{xx}(0) = \frac{1}{N} \sum_{k=0}^{N-1} \quad X^2(k)$$

Energy of finite duration sequences x(n) in terms of frequency components

Linear convolution using DFTs

Let us consider finite duration sequence x(n) 0f length L which excites the Discrete time system having impulse response h(n) of duration M without loss generality, let

$$x(n) = 0 \qquad n < 0 \text{ and } n \geq L$$

$$h(n) = 0 \qquad n < 0 \text{ and } n \geq M$$

y(n) is response of x(n) and it convolution of sum of x(n) and h(n)

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

Duration of y(n) is $N = L + M - 1$

Frequency domain representation of y(n) is Y(ω) = X(ω) $H(\omega)Y(\omega) = X(\omega)H(\omega)$

If y(n) is to be uniquely represented in frequency domain by samples of its spectrum Y(ω) at set of discrete frequencies number of distinct samples must equal or exceed N = L+M-1. Therefore DFT size N≥ L+M-1.is required to represent Y(n) in frequency domain.

Y(ω) at $\omega = \frac{2\pi k}{N}$= X($\omega$) $H(\omega)$ at $\omega = \frac{2\pi k}{N} Y(\omega)|\omega = \frac{2\pi k}{N} = X(\omega)H(\omega)|\omega = \frac{2\pi k}{N}$

$Y(\omega)\ |\omega = \frac{2\pi k}{N} =\ \ X(\omega)\ H(\omega)|\omega = \frac{2\pi k}{N}$

X(k) and H(k) are N point DFTs of corresponding sequences x(n) and h(n) respectively of same duration of Y(k) . if lengths of x(n) and h(n) have duration less than N , pad zeros these sequences with zero s increase their length to N this increase in size does not alter their spectra X(ω) and H(ω) . However by sampling their spectra at N equally spaced samples , increase number of samples that represent these sequences in the frequency domain beyond the minimum number.

Y(k) = X(k) H(k) thus

N point circular convolution of x (n) and h(n) must be equal to linear convolution of x(n) and h(n). Thus zero padding with the DFT can be used to perform linear filtering.

Efficient computation of DFT

To compute the DFT sequence X(k) of N complex valued numbers given another sequence x(n) of duration N according to formula

$$X(k) = \sum_{n=0}^{N-1} x(n)\, {W_N}^{nk} \quad where\ W_N = e^{j\frac{2\pi}{N}kn} \quad for\ k = 0,1,2,3,\ldots\ldots\ldots N-1$$

$$x(n) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\, {W_N}^{-nk} \quad for\ n = 0,1,2,3,----N-1$$

For each value of k direct computation of $X(k)$ involves N complex multiplications (4N real multiplications) and N-1 complex additions (4N-2 real additions)

To evaluate all values of k from 0 to N-1 direct computation of $X(k)$ involves N^2 complex multiplications($4N^2 real\ multiplications$) and N(N-1) complex additions(N(4N-2)) real additions.

Direct computation of N point DFT is basically inefficient because it does not exploit the symmetry and periodic properties of the phase factor W_N.

Direct computation of DFT

If x(n) is complex valued sequence , N point DFT sequence may be expressed as

$$X_R(k) = \sum_{n=0}^{N-1} [x_R(n)\cos(\frac{2\pi}{N}nk) + x_I(n)\sin(\frac{2\pi}{N}nk)]$$

$$X_I(k) = -\sum_{n=0}^{N-1} [x_R(n)\sin(\frac{2\pi}{N}nk) - x_I(n)\cos(\frac{2\pi}{N}nk)]$$

Direct computation of DFT requires

1. $2N^2$ evaluations of trigonometric functions
2. $4N^2$ real multiplications
3. 4N(N-1) real additions
4. A number of indexing and addressing

   Operations 2 and 3 results in the DFT values $X_R(k)\ and\ X_I(k)$. The 3 and 4 operations are necessary to fetch the data x(n) and the phase factors and to store the results,

Periodicity Property

$$W_N{}^{k} = W_N{}^{k+N}$$

$$W_N{}^{k+N} = e^{j\frac{-2\pi(k+N)}{N}}$$

$$= e^{-j\frac{2\pi k}{N}} e^{-j2\pi}$$

$$W_N{}^{k+N} = W_N{}^{k}.\mathbf{1}$$

Symmetry Property

$$W_N^{k+\frac{N}{2}} = W_N^{k}$$

$$= e^{-j\pi} . e^{-j\frac{2\pi k}{N}}$$

$$= -W_N^{k}$$

Radix – 2 FFT algorithms

By adopting the divide – conquer approach for efficient computation of N point DFT is based on the decomposition of an N point DFT into successively smaller DFTs. this basic approach leads to a family of computationally efficient algorithm known as collectively as FFT algorithms . If N can be factorized as product of two integers, that is assumption is that N is not a prime number is not restrictive since we can pad any sequence with zeros to ensure a factorization of form N= L M

N point sequence decompose into L number of M point sequences

If N factorized as N = $r_1, r_2, r_3, r_4 \dots \dots . . r_v$

$$r_1 = r_2 = r_3 = r_4 = \cdots \dots . . r_v = r = r$$

$N = r^v$ this indicate size of DFT is r where number r is called radix r FFT algorithm

If r = 2 radix 2 FFT algorithms

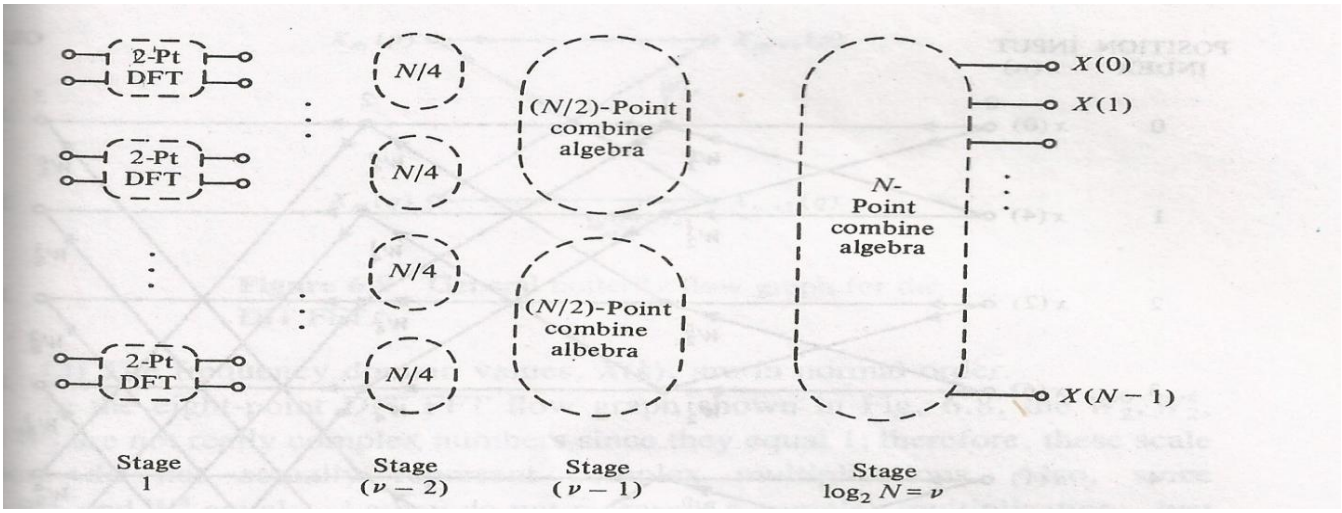
1. Decimation in time (DIT) FFT algorithms.
2. Decimation in frequency (DIF) FFT algorithms.

DIT - FFT algorithms

To decompose N point DFT into successively smaller and smaller number of DFT computations. In this process we exploit both periodicity and symmetry property of complex exponential. Algorithm in which the decomposition is based on decomposition the sequence x (n) into successively smaller subsequences are called decimation in time algorithm.

N is integer power of 2, N = 2^v

In this approach decomposing the N point DFT sequence into two N/2 DFT point sequences, then decomposing N/2 point sequences into two N/4 point DFT sequences and continuing until two point DFT sequence.



Let x(n) is decomposed into two sequences length N/2 one composed of even index value of x(n) and other composed of odd indexed value of x(n).

Input sequence: x (0), x(1) , ……x(N/2 – 1) ……x(N-1)

Even index sequence: x (0) , x(2),x(4) …… x(N-2)

Odd index sequence: x (1), x (3), x (5), …… x (N-1)

N point DFT given by

$$X(k) = \sum_{n=0}^{N-1} x(n)\, W_N^{nk} \qquad for \quad k = 0,1,2,3, \ldots\ldots\ldots N-1$$

$$X(k) = \sum_{n\ even} x(n)\, W_N^{nk} + \sum_{n\ odd} x(n)\, W_N^{nk}$$

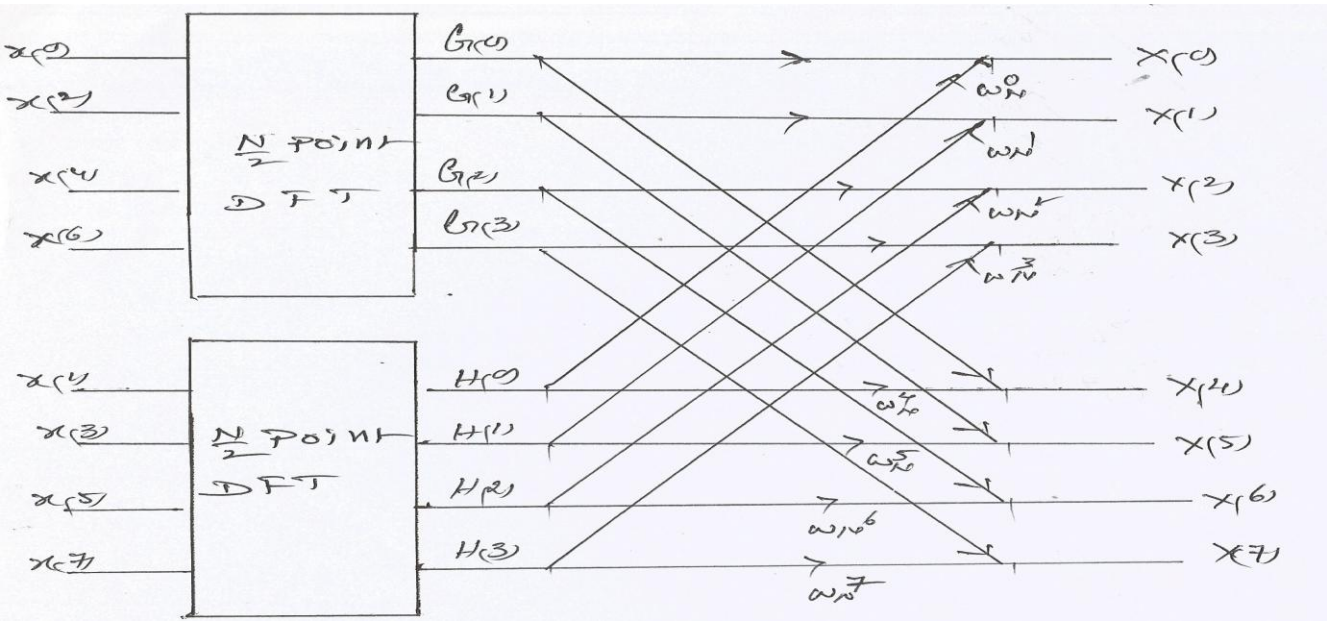
$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)\, W_N^{2rk} + \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)\, W_N^{(2r+1)k}$$

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)\, W_N^{2rk} + W_N^{k} \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)\, W_N^{2rk}$$

$$W_N^{2rk} = W_{\frac{N}{2}}^{k}$$

$$X(k) = \sum_{r=0}^{\frac{N}{2}-1} x(2r)\, W_{\frac{N}{2}}^{kr} + W_N^{\,k} \sum_{r=0}^{\frac{N}{2}-1} x(2r+1)\, W_{\frac{N}{2}}^{kr}$$

$$X(k) = G(k) + W_N^{\,k} H(k)$$



$G(k)$ *is* $\frac{N}{2}$ *point DFT of even indexed sequence and* $H(k)$ *is* $\frac{N}{2}$ *DFT of odd numbered sequence*

G (k) and H (k) are periodic in k with period N/2

G (k) = G (k + N/2): H (k) = H(k + N/2)

N point DFT X (k) is computed by combining of two N/2 point DFTs G (k) and H(k)

Computation involves for N= 8

Number of computations is required to compute two N/2 point DFTs which intern requires $2(N/2)^2$ complex multiplications and approximately $2(N/2)^2$ complex additions. Then two N/2 point are combined, requiring N complex multiplications corresponding to multiplying the second term by $W_N^{\,k}$ and then N complex additions corresponding to adding that product to the first term.

For all values of k requires N + $2(N/2)^2$ complex multiplications and additions. It easy to verify that for N > 2, N + $2(N/2)^2$ will be less than N^2

N + $2(N/2)^2$ = 8 + 16 = 24; $N^2 = 64$

Let us now consider computing each of N/2 point DFTs by breaking each sum into two N/4 point DFTs which would be combined to yield the N/2 point DFTs.

$$G(k) = \sum_{r=0}^{\frac{N}{2}-1} g(r)\, W_{\frac{N}{2}}^{kr} \quad k = 0.1,2,3,4,5,6, \ldots\ldots\ldots\ldots \frac{N}{2}$$

$$G(k) = \sum_{r=0}^{\frac{N}{4}-1} g(2l)\, W_{\frac{N}{2}}^{2kl} + \sum_{r=0}^{\frac{N}{4}-1} g(2l+1)\, W_{\frac{N}{2}}^{k(2l+1)}$$

$$G(k) = \sum_{r=0}^{\frac{N}{4}-1} g(2l)\, W_{\frac{N}{2}}^{2kl} + W_{\frac{N}{2}}^{k} \sum_{r=0}^{\frac{N}{4}-1} g(2l+1)\, W_{\frac{N}{2}}^{2k}$$

$$W_{\frac{N}{2}} = W_N^{\,2}$$

$$G(k) = \sum_{r=0}^{\frac{N}{4}-1} g(2l)\, W_{\frac{N}{2}}^{2kl} + W_N^{\,2k} \sum_{r=0}^{\frac{N}{4}-1} g(2l+1)\, W_{\frac{N}{2}}^{2kl}$$

$$G(k) = \sum_{r=0}^{\frac{N}{4}-1} g(2l)\, W_{\frac{N}{4}}^{kl} + W_N^{\,2k} \sum_{r=0}^{\frac{N}{4}-1} g(2l+1)\, W_{\frac{N}{4}}^{kl}$$

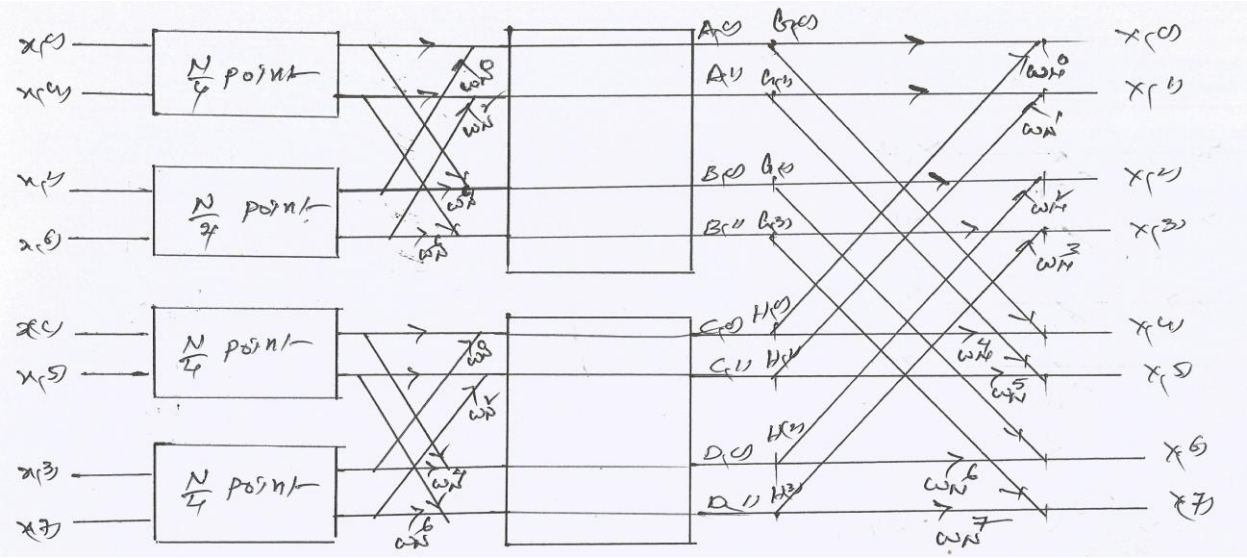
$$G(k) = A(k) + W_N^{\,2k}\ \text{B (k)}$$

N/2 point DFT G (k) is computed by combining of two N/4 point DFTs A (k) and B(k)

Similarly

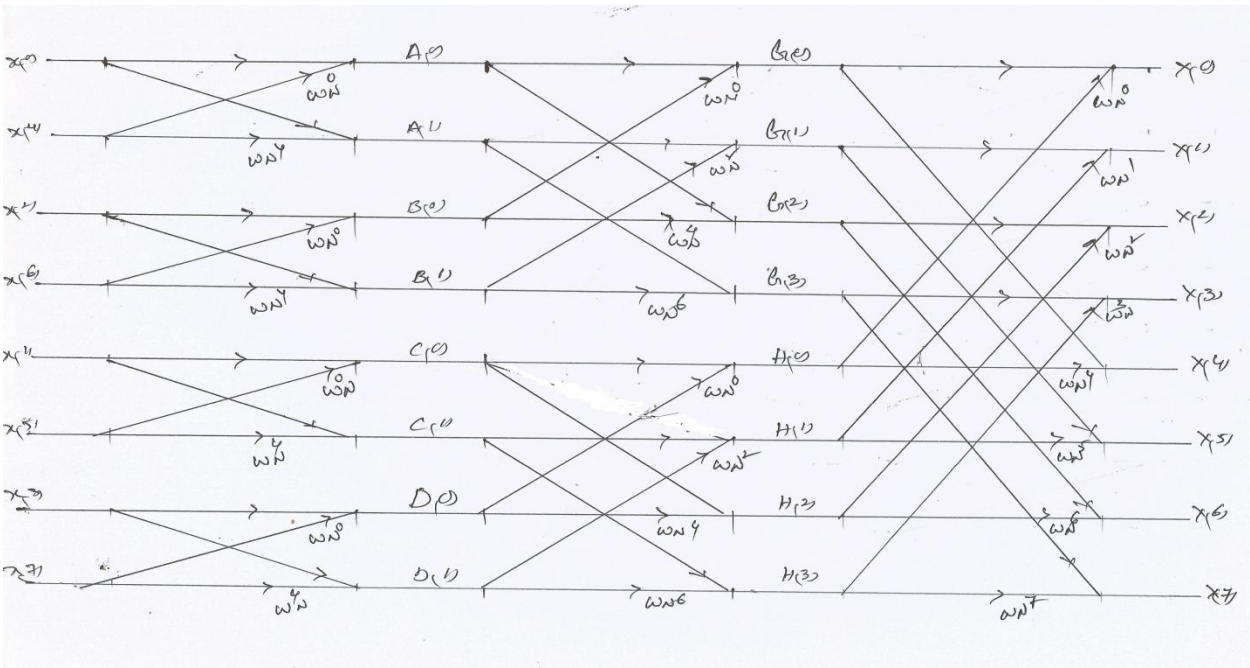
$$H(k) = \sum_{r=0}^{\frac{N}{4}-1} h(2l)\, W_{\frac{N}{4}}^{kl} + W_N^{\,2k} \sum_{r=0}^{\frac{N}{4}-1} h(2l+1)\, W_{\frac{N}{4}}^{kl}$$

$$H(k) = C(k) + W_N^{\,2k}\, D\,(k)$$



N/2 point DFT H (k) is computed by combining of two N/4 point DFTs C (k) and D (k)

When N/2 point DFTs are decomposed into N/4 point DFTs then factor of $(N/2)^2 replaced\ by$ N/2 + $2(N/4)^2$ so over all computation require 2N + $4(N/4)^2$ complex multiplications and complex additions. Further decomposing N/4 point DFT in to N/8 point DFTs and continue until left with two point DFT. In DIT FFT algorithm, input data is shuffled and output data normal order. If N = 2^v this can be at most $v = \log_2 N$ times. So that after carrying out this decomposition as many times as possible number of complex multiplications and additions is equal to N $\log_2 N$.



In place computation

The signal flow describes an algorithm by separating the original sequence into the even numbered and odd numbered points and then continuing to create smaller and smaller subsequences in the same the way. In addition to describing an efficient procedure for computing the DFT also suggests useful way of storing the original data and storing results of computation in the intermediate arrays.

According to signal flow graph, each stage of computation takes a set of N complex numbers and transforms them into another set of N complex numbers. This process repeated $v = \log_2 N$ times resulting computation of DFT. when we are going to implement computations we need to use two arrays of storage registers , one for array being computed and one for the data being used in the computation. One set of storage registers would contain the input data and second set of storage registers would contain the computed results for the first stage. While validity of fig in not tied to order in which input data are stored, let us order set of complex numbers in the same order that they appear in fig

$X_m(l)\ as$ Input array and $X_{m+1}(l)$ as output array for (m+1)st stage computation

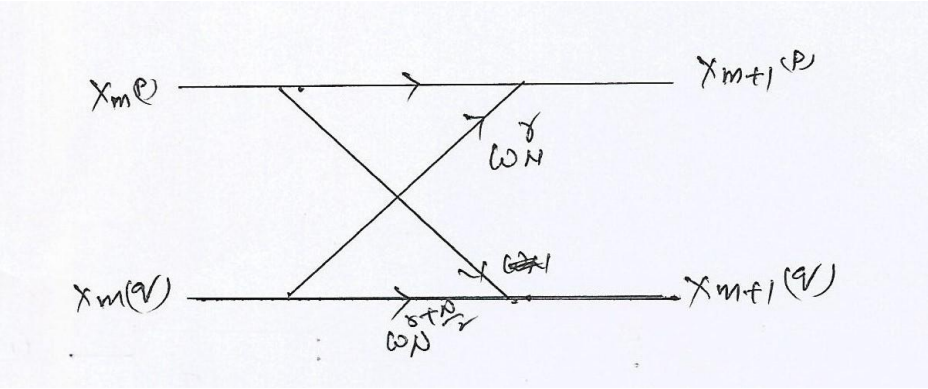
The basic computation in signal flow graph referred to as butterfly computation

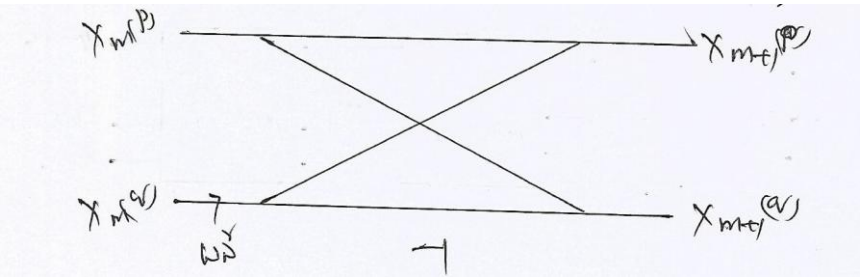
$$X_{m+1}(p) = X_m(p) + W_N{}^r X_m(q)$$

$$X_{m+1}(q) = X_m(p) + W_N{}^{r+\frac{N}{2}} X_m(q)$$

$$W_N{}^{\frac{N}{2}} = e^{-(j\frac{2\pi}{N})\frac{N}{2}} = e^{-j\pi} = -1$$

$$X_{m+1}(q) = X_m(p) - W_N{}^r X_m(q)$$

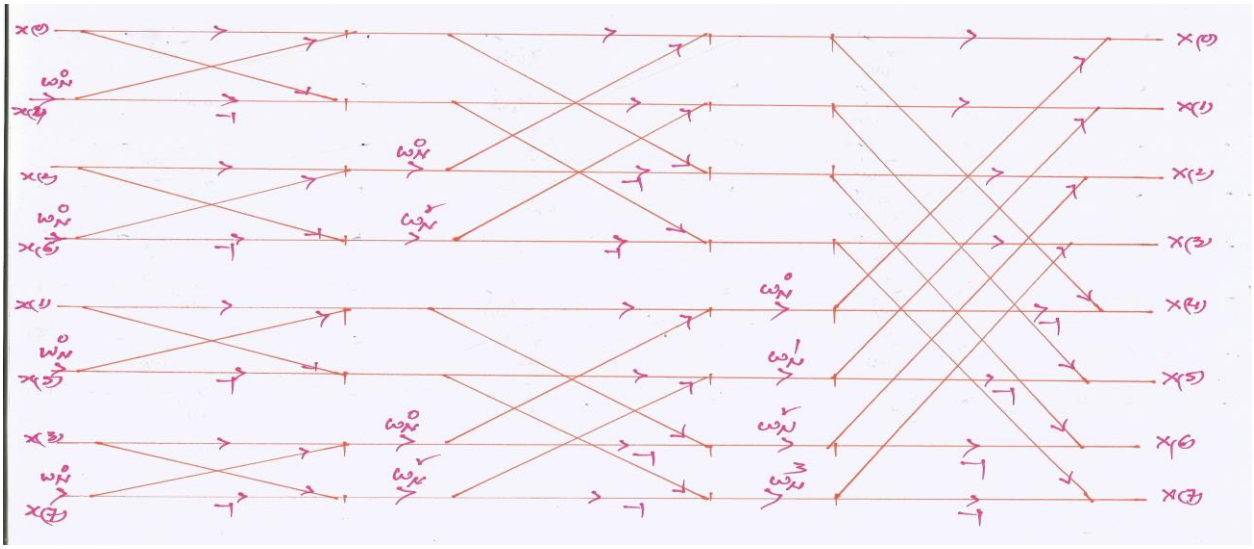




Using above equations number complex multiplications reduced by factor of 2

Number of butterflies required per stage is N/2

Total number of complex multiplications $\frac{N}{2}\log_2 N$



P, q and r vary from stage to stage in manner such that complex numbers in locations p,q of mth array are required to compute the elements p and q of the (m+1)st array. Thus only one complex array of N storage registers is physically necessary to implement the complete computation if $X_{m+1}(p)$ and $X_{m+1}(q)$ are stored in same registers as $X_m(p)$ and $X_m(q)$ respectively this kind of computation is commonly referred as in place computation, since it has advantage that as a new array is computed the results can be stored in the same storage locations as original array. signal flow graph represent an in place computation is tied to the fact that associate nodes in the flow graph that are on the same horizontal line with the same storage location and that computation between two arrays consists of butterfly computation in which the input nodes and output nodes are horizontal adjacent. Input data stored in no sequential order in which the input data are stored is in bit reversal order. If $(n_2n_1n_0)$ is binary representation of index of sequence x(n) then the sequence value x$(n_2n_1n_0)$ is stored in the array position $X_0(n_0n_1n_2)$

Bit reversal order necessary for in place computation ordering of the sequence x (n) in such as manner that the DFT computation is decomposed into successively smaller DFT computations. It is a process to decompose input sequence x(n) divided into even numbered samples and odd numbered samples, with even numbered samples occurring in the top half and odd number samples occurring bottom half . Such separation of data carried out by examining the least significant but n_0 in the index. If LSB is zero, the sequence value corresponds to an even numbered sample and therefore will appear in the top half and if LSB is one , the sequence value corresponds to odd numbered sequence and will appear in the bottom half the array. Next even and odd subsequences are each sorted in their even and odd parts and this can be done by examining the second least significant bit in the data index. in second LSB is 0 the sub sequence value correspond s to even numbered sample and therefore will appear top half , if second LSB is one the sub sequence value corresponds to odd numbered sample and will appear bottom half,. This process repeated until N subsequences of length 1 are obtained. This sorting even and odd indexed subsequences is shown in fig .

Decimation in Frequency (DIF) FFT

Output sequence X (k) into smaller and smaller number of sequences in the same manner as DIT, these classes of FFTs based on the procedure referred to as decimation in frequency (DIF). To derive the DIF FFT for N is always power of 2, we can first divide the input sequence into the first half and the last half of the points.

Input sequence: x (0), x(1) , ……x(N/2 – 1), x(N/2) ……x(N-1)

First half sequence: x (0) , x(1),x(2) …… x($\frac{N}{2}-1$)

Second half sequence: x ($\frac{N}{2}$), x ($\frac{N}{2}+1$), …… x (N-1)

$$X(k)=\sum_{n=0}^{N-1}x(n)\,W_N{}^{nk} \qquad for \quad k=0,1,2,3,\ldots\ldots\ldots N-1$$

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

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

$$W_N{}^{(\frac{N}{2})k}=e^{-(j\frac{2\pi}{N})\frac{Nk}{2}}=e^{-j\pi k}=(-1)^k$$

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

K is even number say k = 2r

$$X(2r) = \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) + x\left(n+\frac{N}{2}\right)\right] W_N{}^{2nr}$$

$$X(2r) = \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) + x\left(n+\frac{N}{2}\right)\right] W_{\frac{N}{2}}{}^{nr} \qquad r = 0,1,2,\ldots\ldots\ldots\ldots\ldots\ldots\ldots\frac{N}{2}-1$$

K is odd number say k = 2r+1

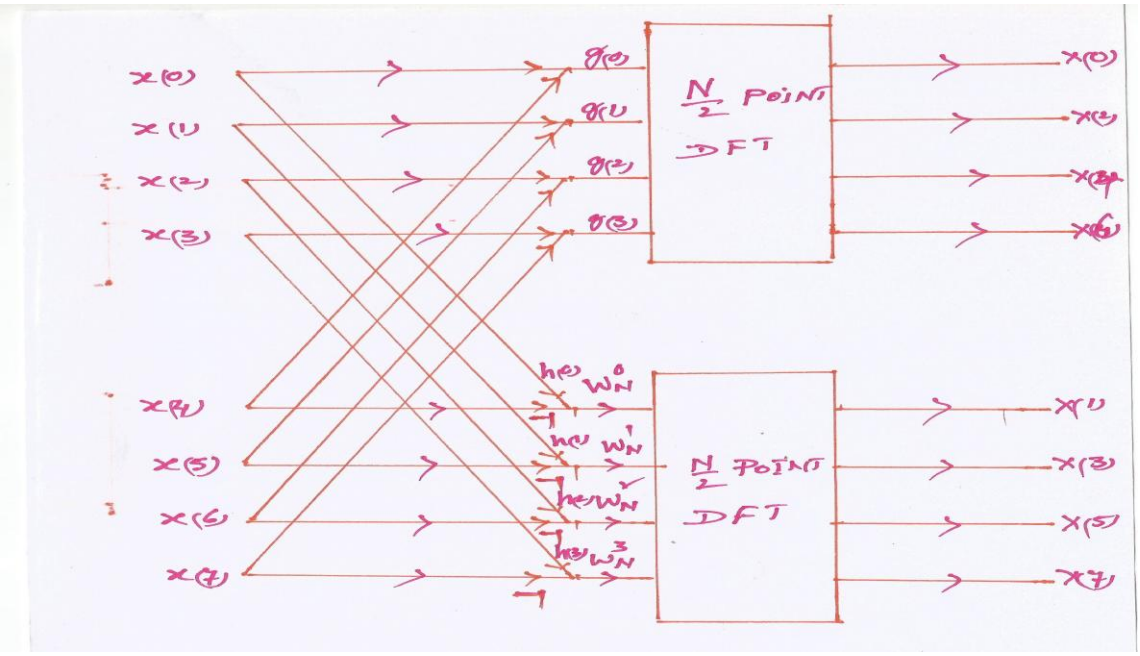
$$X(2r+1) = \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) - x\left(n+\frac{N}{2}\right)\right] \;\; W_N{}^{n(2r+1)}$$

$$X(2r+1) = \sum_{n=0}^{\frac{N}{2}-1} \left[x(n) - x\left(n+\frac{N}{2}\right)\right] \;\; W_N{}^{n}\; W_N{}^{2nr}$$

$$X(2r+1) = \sum_{n=0}^{\frac{N}{2}-1} \left\{\left[x(n) - x\left(n+\frac{N}{2}\right)\right] \;\; W_N{}^{n}\right\} W_{\frac{N}{2}}{}^{nr} \quad r = 0,1,2,\ldots\ldots\ldots\ldots\ldots\ldots\frac{N}{2}-1$$

Assume $g(n) = x(n) + x\left(n+\frac{N}{2}\right)$ and $h(n) = \left[x(n) - x\left(n+\frac{N}{2}\right)\right] \;\; W_N{}^{n}$

G (k) = $\sum_{n=0}^{\frac{N}{2}-1} g(n)\; W_{\frac{N}{2}}{}^{nk}$ and $H(k) = \sum_{n=0}^{\frac{N}{2}-1} h(n)\; W_{\frac{N}{2}}{}^{nk}$



N/2 point DFT can be computed by computing the even numbered and odd numbered output points for those DFTs separately .similar way, N/4 point DFT can be computed by computing the even numbered and odd numbered output points for those DFTs separately. This process will continue until two point DFT.

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

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

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

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

$$W_{\frac{N}{2}}^{\frac{N}{4}k} = e^{-(j\frac{4\pi}{N})\frac{Nk}{4}} = (-1)^k$$

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

K is even

$$G(2l) = \sum_{n=0}^{\frac{N}{4}-1} \left[g(n) + g\left(n+\frac{N}{4}\right)\right] W_{\frac{N}{2}}^{2nl}$$

$$G(2l) = \sum_{n=0}^{\frac{N}{4}-1} \left[g(n) + g\left(n+\frac{N}{4}\right)\right] W_{\frac{N}{4}}^{nl} \quad l = 0,1,2 \ldots\ldots\ldots\ldots\ldots \frac{N}{4}-1.$$

K is odd that is k = 2l+1

$$G(2l+1) = \sum_{n=0}^{\frac{N}{4}-1} \left\{\left[g(n) - g\left(n+\frac{N}{4}\right)\right] W_{\frac{N}{2}}^{n}\right\} W_{\frac{N}{4}}^{nl} \quad l = 0,1,2 \ldots\ldots\ldots\ldots\ldots \frac{N}{4}-1.$$

$$a(n) = g(n) + g\left(n+\tfrac{N}{4}\right) \text{ and } b(n) = \left[g(n) - g\left(n+\tfrac{N}{4}\right)\right] W_{\frac{N}{2}}^{n}$$

A(k) = G(2l) B(k) = G(2l+1)

Similarly

$$H(k) = \sum_{n=0}^{\frac{N}{2}-1} h(n)\, W_{\frac{N}{2}}^{nk}$$

$$H(2l) = \sum_{n=0}^{\frac{N}{4}-1} \left[h(n) + h\left(n+\frac{N}{4}\right)\right] W_{\frac{N}{4}}^{nl} \quad l = 0,1,2 \ldots\ldots\ldots\ldots\ldots \frac{N}{4}-1.$$

$$H(2l+1) = \sum_{n=0}^{\frac{N}{4}-1} \left\{\left[h(n) - h\left(n+\frac{N}{4}\right)\right] W_{\frac{N}{2}}^{n}\right\} W_{\frac{N}{4}}^{nl} \quad l = 0,1,2 \ldots\ldots\ldots\ldots\ldots \frac{N}{4}-1.$$

$$c(n) = h(n) + h\left(n+\tfrac{N}{4}\right) \text{ And } d(n) = \left[h(n) - h\left(n+\tfrac{N}{4}\right)\right] W_{\frac{N}{2}}^{n}$$

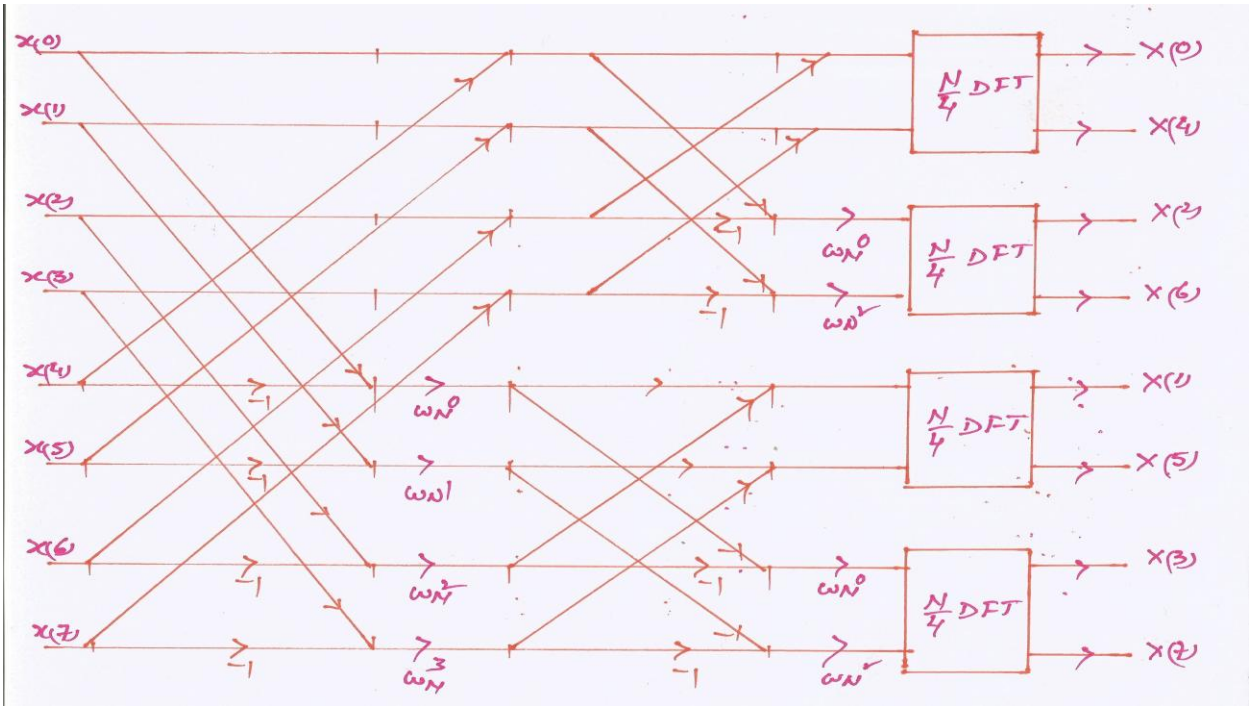
(k) = G(2l) D(k) = G(2l+1)

$$A(k) = \sum_{n=0}^{\frac{N}{4}-1} a(n)\ W_{\frac{N}{4}}^{nk} \quad k = 0,1,2 \ldots\ldots\ldots\ldots \frac{N}{4} - 1.$$

$$B(k) = \sum_{n=0}^{\frac{N}{4}-1} b(n)\ W_{\frac{N}{4}}^{n} \quad k = 0,1,2 \ldots\ldots\ldots\ldots \frac{N}{4} - 1.$$

$$C(k) = \sum_{n=0}^{\frac{N}{4}-1} a(n)\ W_{\frac{N}{4}}^{nk} \quad k = 0,1,2 \ldots\ldots\ldots\ldots \frac{N}{4} - 1.$$

$$D(k) = \sum_{n=0}^{\frac{N}{4}-1} b(n)\ W_{\frac{N}{4}}^{n} \quad k = 0,1,2 \ldots\ldots\ldots\ldots \frac{N}{4} - 1.$$



DIF FFT algorithm requires $\frac{N}{2}\log_2 N$complex multiplications $and\ N\log_2 N\ \ complex\ additions$ and input sequence is normal order and output sequence if shuffled order.

Butterfly computations

$$X_{m+1}(p) = X_m(p) + \ X_m(q)$$

$$X_{m+1}(q) = [X_m(p) - \ X_m(q)]W_N{}^r$$

Computation of inverse DFT

$$x(n) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)\, W_N{}^{-nk} \quad for\ n = 0,1,2,3,- - - - N-1$$

Comparing to DFT computational procedure remains same expect that twiddle factors are negative power of W_N and output must be scaled by 1/N therefore, an inverse Fast Fourier Transform (IFFT) flow diagram can be obtained from FFT flow diagram by replacing the x(n) s by X(k) , scaling input data by 1/N a per stage factor of ½ when N is power of 2 and changing the exponents of $W_N\ to$ negative values .

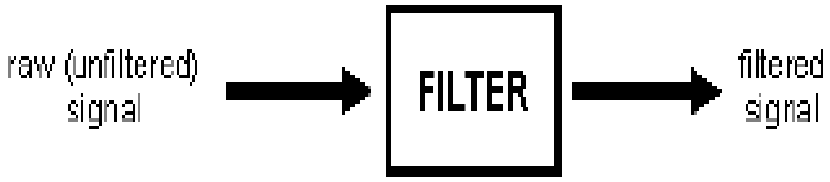
UNIT-III

STRUCTURE OF IIR FILTERS

Analog Filters:

Analog and digital filters

In signal processing, the function of a *filter* is to remove unwanted parts of the signal, such as random noise, or to extract useful parts of the signal, such as the components lying within a certain frequency range. The following block diagram illustrates the basic idea.



There are two main kinds of filter, analog and digital. They are quite different in their physical makeup and in how they work. An **analog** filter uses analog electronic circuits made up from components such as resistors, capacitors and opamps to produce the required filtering effect. Such filter circuits are widely used in such applications as noise reduction, video signal enhancement, graphic equalizers in hi-fi systems, and many other areas. There are well-established standard techniques for designing an analog filter circuit for a given requirement.

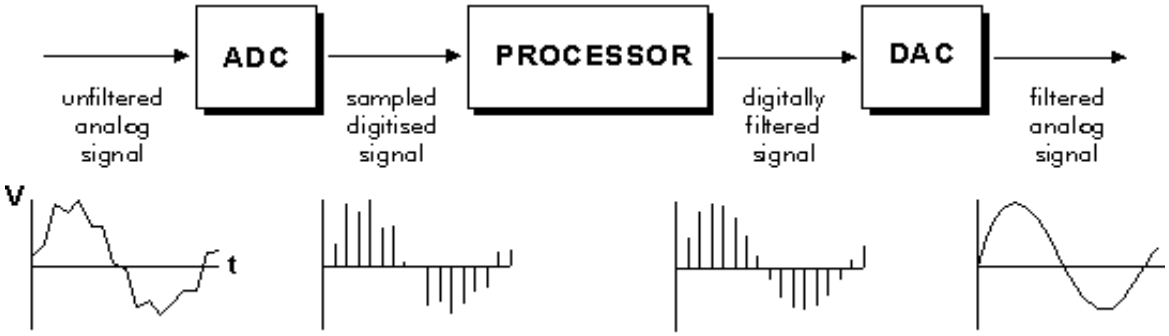
At all stages, the signal being filtered is an electrical voltage or current which is the direct analogue of the physical quantity (e.g. a sound or video signal or transducer output) involved.

A **digital** filter uses a digital processor to perform numerical calculations on sampled values of the signal. The processor may be a general-purpose computer such as a PC, or a specialized DSP (Digital Signal Processor) chip.

The analog input signal must first be sampled and digitized using an ADC (analog to digital converter). The resulting binary numbers, representing successive sampled values of the input signal, are transferred to the processor, which carries out numerical calculations on them. These calculations typically involve multiplying the input values by constants and adding the products together. If necessary, the results of these calculations, which now represent sampled values of the filtered signal, are output through a DAC (digital to analog converter) to convert the signal back to analog form.

Note that in a digital filter, the signal is represented by a sequence of numbers, rather than a voltage or current.

The following diagram shows the basic setup of such a system.



Define

$$R(s) := H_a(s) \cdot H_a(-s).$$

Then

$$R(j\omega) = |H_a(j\omega)|^2.$$

For convenience, define

$$F(\omega) := R(j\omega),$$

so that we do not need to carry j along. Then

$$\boxed{R(s) = F(s/j)}$$

and

$$\boxed{F(\omega) = |H_a(j\omega)|^2.}$$

Note that:

$$R(-s) = R(s)$$

so $R(s)$ is an *even* function of s. Also note that,

$$F(-\omega) = R(-j\omega) = R(j\omega) = F(\omega)$$

so $F(\omega)$ is an *even* function of ω.

Example: If

$$H_a(s) = \frac{0.2\,s^2 + 0.02\,s + 1}{s^3 + s^2 + 2\,s + 1}$$

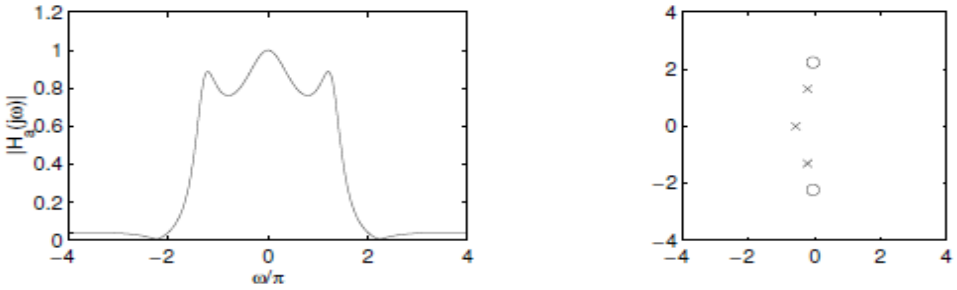
then

$$R(s) = H_a(s) \cdot H_a(-s) = \frac{0.04\,s^4 + 0.3996\,s^2 + 1}{-s^6 - 3\,s^4 - 2\,s^2 + 1}$$

and

$$F(\omega) = R(j\omega) = \frac{0.04\,\omega^4 - 0.3996\,\omega^2 + 1}{\omega^6 - 3\,\omega^4 + 2\,\omega^2 + 1}$$

Note that the function $F(\omega)$ contains only even powers of ω. $F(\omega)$ is an even function of ω. The magnitude of the frequency response and the pole-zero diagram of $H_a(s)$ is shown in the figure.



The classical analog filters are developed by the following approach. Design the function $F(\omega)$ so that $R(s) = F(s/j)$ can be spectrally factored, then compute a spectral factor to obtain $H_a(s)$. That means, given $R(s)$, find $H_a(s)$ so that

$$R(s) = H_a(s) \cdot H_a(-s).$$

This is the same approach taken in the design of minimum-phase FIR filters where the first step was to design a linear-phase filter with a non-negative frequency response and the second step entails a spectral factorization. This leads to the following problem.
Design a rational function $F(\omega) = |H_a(j\omega)|^2$ such that

1. $F(\omega) \geq 0$
2. $F(\omega)$ is an *even* function of ω.

The classical analog lowpass filters are developed by writing $F(\omega)$ as

$$\boxed{F(\omega) = \frac{1}{1 + \epsilon^2 V(\omega)^2}.}$$

Note that whatever $V(\omega)$ is, $F(\omega)$ will be non-negative. The non-negativity of $F(\omega)$ is built into this expression. $V(\omega)$ does not need to be a non-negative function. Note that if $V(\omega)$ is a rational function, then $F(\omega)$ will also be a rational function. The approach is to design a rational function $V(\omega)$. Note that

1. when $V(\omega)$ is very large, $F(\omega)$ is close to 0.
2. when $V(\omega)$ is very small, $F(\omega)$ is close to 1.

So in the pass-band, $V(\omega)$ should be small, and in the stop-band, $V(\omega)$ should be large.

THE BUTTERWORTH ANALOG FILTER

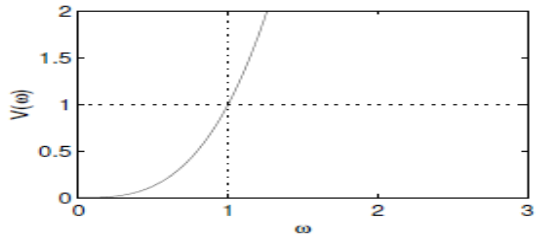
The simplest case is the Butterworth filter. For the Butterworth filter,

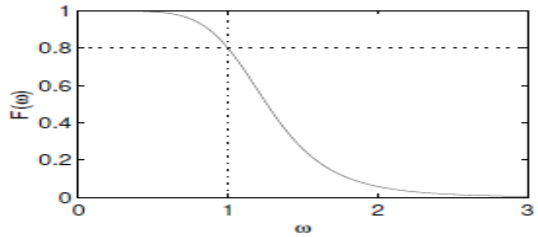
$$V(\omega) = \omega^N. \tag{40}$$

Then

$$\boxed{F(\omega) = \frac{1}{1 + \epsilon^2\,\omega^{2N}}.} \tag{41}$$

For example, for $N = 3$ and $\epsilon = 0.5$, the functions $V(\omega)$ and $F(\omega)$ are as shown.





Note that the value of $F(\omega)$ at the frequency $\omega = 1$ is

$$F(1) = \frac{1}{1 + \epsilon^2}. \tag{42}$$

For example, when $\epsilon = 0.5$, then

$$F(1) = 1/(1 + 0.5^2) = 1/(1 + 1/4) = 1/(5/4) = 0.8. \tag{43}$$

This is indicated by the dotted line in the figure above. Note that $F(\omega)$ is *monotonic* in both the pass-band and the stop-band.

To obtain the transfer function $H_a(s)$, we can first obtain $R(s)$ by,

$$R(s) = F(s/j) = \frac{1}{1+\epsilon^2(s/j)^{2N}} \tag{44}$$

$$= \frac{1}{1+\epsilon^2(s^2/j^2)^N} \tag{45}$$

$$= \frac{1}{1+\epsilon^2(-s^2)^N} \tag{46}$$

To find the poles, set the denominator of $R(s)$ to zero. This will give the roots of $Q(s) \cdot Q(-s)$. Once we find these roots, we can identify the roots of $Q(s)$ by taking those in LHP.

$$1+\epsilon^2(-1)^N s^{2N} = 0 \tag{47}$$

$$s^{2N} = \frac{-1}{\epsilon^2(-1)^N} \tag{48}$$

$$s^{2N} = \frac{-1(-1)^N}{\epsilon^2} \tag{49}$$

When N is **even** this becomes

$$s^{2N} = \frac{-1}{\epsilon^2}. \tag{50}$$

To find an expression for the poles, we can write

$$-1 = e^{j\pi} \tag{51}$$

or

$$-1 = e^{j(\pi+2\pi k)}. \tag{52}$$

This leads to the following chain

$$s^{2N} = \frac{-1}{\epsilon^2} \tag{53}$$

$$= \frac{e^{j(\pi+2\pi k)}}{\epsilon^2} \tag{54}$$

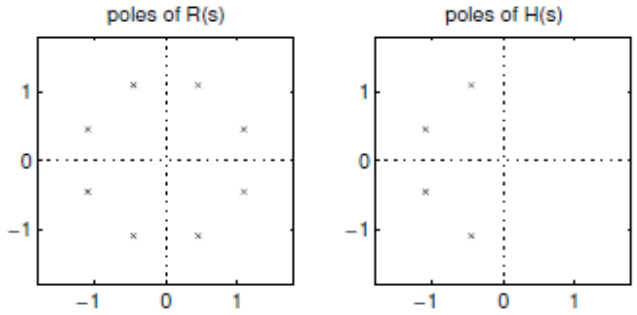
$$s = \left(\frac{e^{j(\pi+2\pi k)}}{\epsilon^2}\right)^{\frac{1}{2N}} \tag{55}$$

$$s = \frac{e^{j(\pi+2\pi k)/(2N)}}{\epsilon^{1/N}} \tag{56}$$

$$s = \frac{e^{j(1+2k)\frac{\pi}{2N}}}{\epsilon^{1/N}} \tag{57}$$

for $0 \leq k \leq 2N-1$.

For example, when $N = 4$, the poles are indicated by the marks in the figure, they are equally spaced on a circle of radius $\frac{1}{\epsilon^{1/N}}$.



When N is **odd**, setting the denominator of $R(s)$ to zero gives

$$s^{2N} = \frac{1}{\epsilon^2}. \tag{58}$$

To find an expression for the poles, we can write

$$1 = e^{j2\pi} \tag{59}$$

or

$$1 = e^{j2\pi k}. \tag{60}$$

Adding an integer multiple of 2π in the exponent leads to the following chain

$$s^{2N} = \frac{1}{\epsilon^2} \tag{61}$$

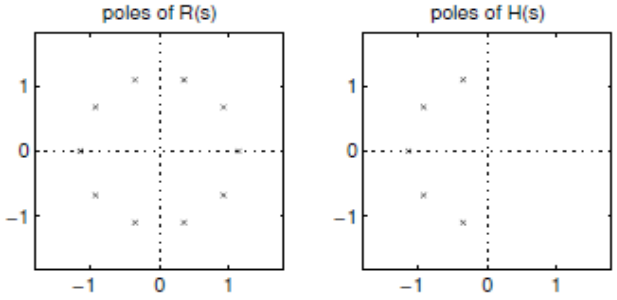
$$= \frac{e^{j2\pi k}}{\epsilon^2} \tag{62}$$

$$s = \left(\frac{e^{j2\pi k}}{\epsilon^2}\right)^{\frac{1}{2N}} \tag{63}$$

$$s = \frac{e^{j\pi k/N}}{\epsilon^{1/N}} \tag{64}$$

for $0 \leq k \leq 2N-1$.

For example, when $N = 5$, the poles are indicated by the marks in the figure, they are equally spaced on a circle of radius $\frac{1}{\epsilon^{1/N}}$.



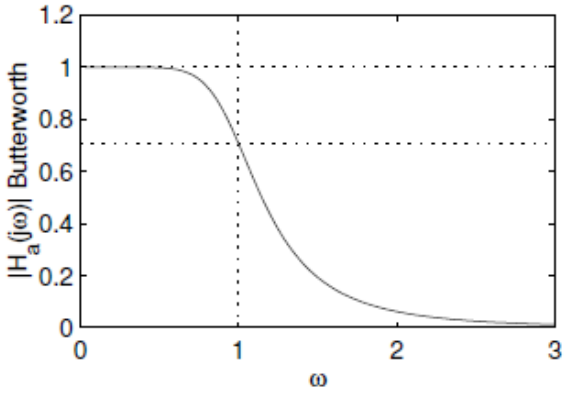
The roots in the LHP, are given by the following formula, valid when N is either even or odd.

$$s_k = \frac{1}{\epsilon^{1/N}} \cdot e^{j(2k+1+N)\pi/(2N)} \tag{65}$$

for $0 \leq k \leq N-1$. These are the poles of the analog Butterworth filter of order N. The numerator is simply 1 so there are no finite zeros.

The Matlab command buttap can be used to obtain the poles of the Butterworth filter of degree N. (This command is part of the Matlab *Signal Processing Toolbox*.) The only input to this command is the filter order N. The result of buttap is normalized so that at $\omega = 1$, $|H_a(j\omega)| = 1/\sqrt{2}$.
The following code uses buttap to obtain the transfer function of a fourth order Butterworth analog filter and plots $|H_a(j\omega)|$.

```
N = 4;
[z,p,k] = buttap(N);
P = k*poly(z);
Q = poly(p);
w = 0:0.01:3;
H = polyval(P,j*w)./polyval(Q,j*w);
figure(1)
clf
subplot(4,2,1)
plot(w,abs(H),[0 3],[1 1]/sqrt(2),':',[1 1],[0 1.2],':',[0 3],[1 1],'
axis([0 3 0 1.2])
xlabel('\omega')
ylabel('|H_a(j\omega)| Butterworth')

orient tall
print -deps butex2
```



THE CHEBYSHEV-I ANALOG FILTER

The Chebyshev-I analog filter is based on the Chebyshev polynomials $C_N(\omega)$. For the Chebyshev-I filter,

$$V(\omega) = C_N(\omega). \tag{66}$$

Then

$$\boxed{F(\omega) = \frac{1}{1 + \epsilon^2\, C_N(\omega)^2}.} \tag{67}$$

The remarkable Chebyshev polynomials can be generated by the following recursive formula.

$$C_0(\omega) := 1 \tag{68}$$

$$C_1(\omega) := \omega \tag{69}$$

$$C_{k+1}(\omega) := 2\,\omega\, C_k(\omega) - C_{k-1}(\omega). \tag{70}$$

The next few $C_k(\omega)$ are

$$C_2(\omega) = 2\,\omega^2 - 1 \tag{71}$$

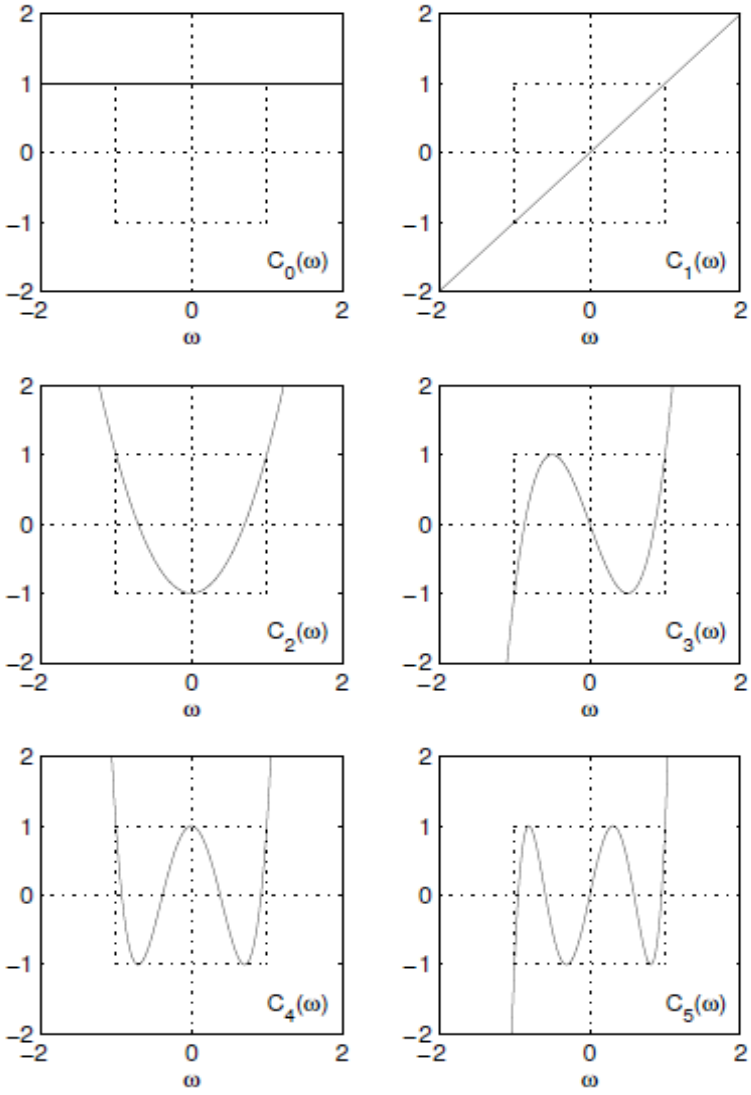
$$C_3(\omega) = 4\,\omega^3 - 3\,\omega \tag{72}$$

$$C_4(\omega) = 8\,\omega^4 - 8\,\omega^2 + 1 \tag{73}$$

$$C_5(\omega) = 16\,\omega^5 - 20\,\omega^3 + 5\,\omega. \tag{74}$$

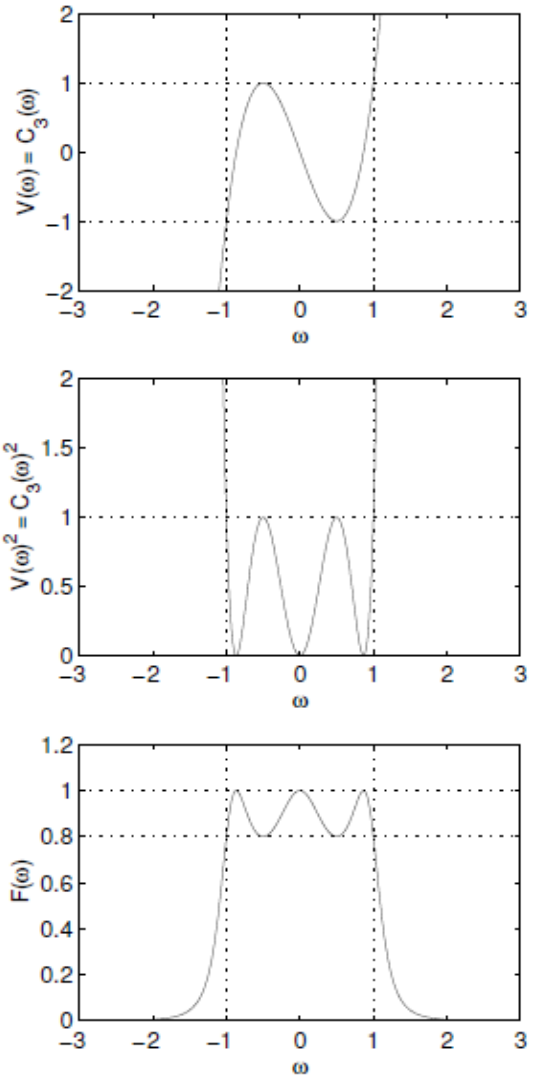
As can be seen in the following figures, in the interval $-1 \le \omega \le 1$, the Chebyshev polynomial oscillates between -1 and 1. This will create a equiripple behavior in the pass-band of the resulting analog filter.

CHEBYSHEV POLYNOMIALS



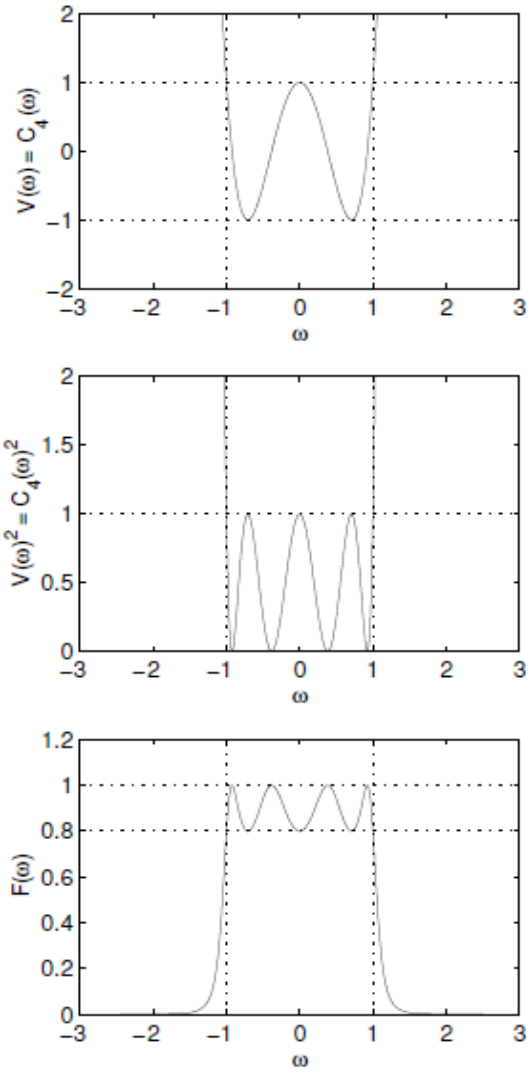
THE CHEBYSHEV-I ANALOG FILTER

For example, when $N = 3$ and $\epsilon = 0.5$, then the functions $V(\omega)$ and $F(\omega)$ are as shown.



THE CHEBYSHEV-I ANALOG FILTER

When $N = 4$ and $\epsilon = 0.5$, then the functions $V(\omega)$ and $F(\omega)$ are as shown.



The Chebyshev-I analog filter has no finite zeros.

Skipping the derivation, the poles of the Chebyshev-I filter lie at

$$s_k = -\sinh(v)\,\sin\left(\frac{(2k+1)\pi}{2N}\right) + j\,\cosh(v)\,\cos\left(\frac{(2k+1)\pi}{2N}\right) \tag{75}$$

for $0 \leq k \leq N-1$ where

$$v = \frac{\sinh^{-1}(1/\epsilon)}{N}. \tag{76}$$

THE CHEBYSHEV-I ANALOG FILTER

The frequency response of the Chebyshev-I analog filter is equiripple in the pass-band, and monotonic in the stop-band.

The Matlab command `cheb1ap` can be used to obtain the poles of the Chebyshev-I filter of degree N. The input arguments of this command are the filter degree N and the pass-band ripple size R_p in dB. For this filter $|H_a(j\omega)|$ lies between the bounds 1 and $1-\delta_p$ in the pass-band. The value R_p is related to δ_p by

$$1 - \delta_p = 10^{-R_p/20} \tag{77}$$

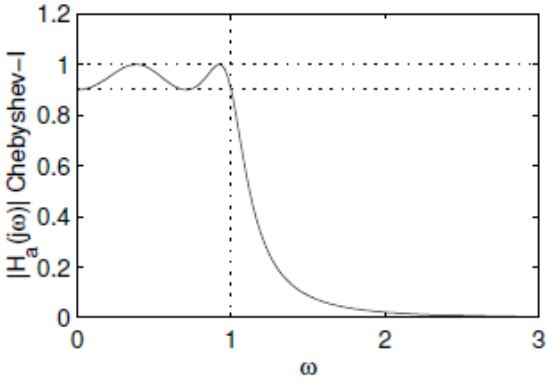
or

$$R_p = -20 \log_{10}(1 - \delta_p). \tag{78}$$

For example, to obtain a Chebyshev-I filter of degree N such that $|H_a(j\omega)|$ lies between 0.9 and 1 in the pass-band, we can use `cheb1ap` with

$$R_p = -20 \log_{10}(0.9). \tag{79}$$

The code on the next page uses `cheb1ap` to obtain the transfer function of a fourth order Chebyshev-I analog filter and plots $|H_a(j\omega)|$.

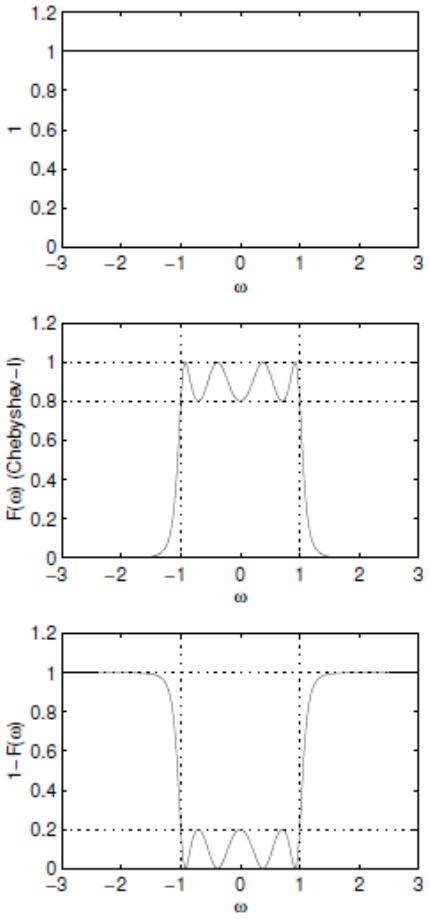


THE CHEBYSHEV-I ANALOG FILTER

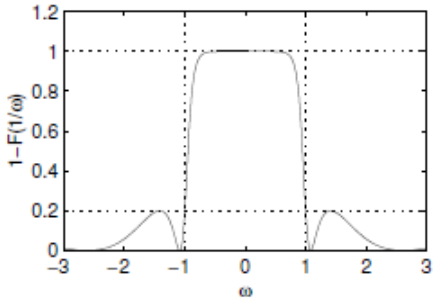
```
N  = 4;
dp = 0.1;
Rp = -20*log10(1-dp);
[z,p,k] = cheb1ap(N,Rp);
P = k*poly(z);
Q = poly(p);
w = 0:0.01:3;
H = polyval(P,j*w)./polyval(Q,j*w);
figure(1)
clf
subplot(4,2,1)
plot(w,abs(H),[0 3],[1 1]*(1-dp),':',[1 1],[0 1.2],':',[0 3],[1 1],':');
axis([0 3 0 1.2])
xlabel('\omega')
ylabel('|H_a(j\omega)| Chebyshev-I')

orient tall
print -deps chebex2
```

THE CHEBYSHEV-II ANALOG FILTER

The Chebyshev-II analog filter (also called the inverse-Chebyshev filter) is designed so as to have a monotonic pass-band and an equiripple stop-band. This type of frequency response can be obtained by a two step procedure. First, subtract the Chebyshev-I $F(\omega)$ from 1. This is illustrated in the following figure.



Second, perform the change of variables $\omega \leftarrow 1/\omega$. This results in the Chebyshev-II frequency response.



As the figure shows, the frequency response has a monotonic pass-band and an equiripple stop-band. According to the two step procedure, the formula for $F(\omega)$ for the Chebyshev-II filter is given by

$$F(\omega) = 1 - \frac{1}{1 + \epsilon^2\, C_N(1/\omega)^2} \tag{80}$$

or

$$\boxed{F(\omega) = \frac{\epsilon^2\, C_N(1/\omega)^2}{1 + \epsilon^2\, C_N(1/\omega)^2}.} \tag{81}$$

The Chebyshev-II filter has zeros as the numerator is not just 1. It turns out that all of its zeros lie on the imaginary axis. Skipping the details, the zeros are given by

$$z_k = \frac{j}{\cos((2k+1)\pi/(2N))} \tag{82}$$

for $0 \leq k \leq N-1$. Surprisingly, the poles of the Chebyshev-II analog filter are the reciprocals of the poles of the Chebyshev-I analog filter.

THE CHEBYSHEV-II ANALOG FILTER

The Matlab command cheb2ap can be used to obtain the poles and zeros of the Chebyshev-II filter of degree N. The input arguments of this command is the filter degree N and the stop-band ripple size R_s in dB. For this filter $|H_a(j\omega)|$ lies between the bounds 0 and δ_s in the stop-band. The value R_s is related to δ_s by

$$\delta_s = 10^{-R_s/20} \tag{83}$$

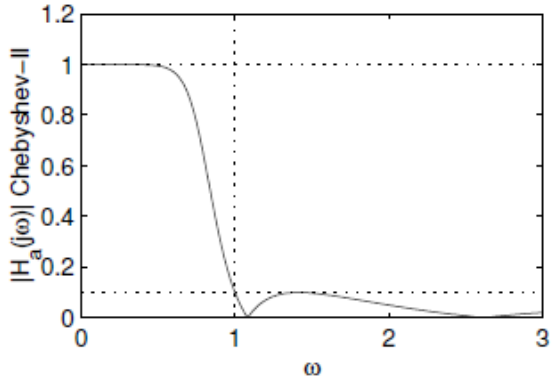
or

$$R_s = -20\ \log_{10}(\delta_s). \tag{84}$$

For example, to obtain a Chebyshev-II filter of degree N such that $|H_a(j\omega)|$ lies between 0 and 0.1 in the stop-band, we can use cheb2ap with

$$R_s = -20\ \log_{10}(0.1). \tag{85}$$

The code on the next pages uses cheb2ap to obtain the transfer function of a fourth order Chebyshev-II analog filter and plots $|H_a(j\omega)|$.



THE CHEBYSHEV-II ANALOG FILTER

```
N  = 4;
ds = 0.1;
Rs = -20*log10(ds);
[z,p,k] = cheb2ap(N,Rs);
P = k*poly(z);
Q = poly(p);
w = 0:0.01:3;
H = polyval(P,j*w)./polyval(Q,j*w);
figure(1)
clf
subplot(4,2,1)
plot(w,abs(H),[0 3],[1 1]*ds,':',[1 1],[0 1.2],':',[0 3],[1 1],':');
axis([0 3 0 1.2])
xlabel('\omega')
ylabel('|H_a(j\omega)| Chebyshev-II')

orient tall
print -deps ichebex2
```

Design of IIR Filters from Analog Filters

Why IIR? With IIR designs we can get the same approximation accuracy (of the magnitude response) as FIR but with a lower order filter. The tradeoff is nonlinear phase.

Analog filter design is a mature field. There are well known methods for selecting RLC combinations to approximate some desired frequency response $H_d(F)$.

Generally, the more (passive) components used, the closer one can approximate $H_d(F)$, (to within component tolerances).

Essentially *all* analog filters are IIR, since the solutions to linear differential equations involve infinite-duration terms of the form $t^m \, \mathrm{e}^{\lambda t} \, u(t)$.

One way to design IIR digital filters is to piggyback on this wealth of design experience for analog filters.

What do analog filters look like?

Any RLC network is describe by a **linear constant coefficient differential equation** of the form

$$\sum_{k=0}^{N} \alpha_k \frac{d^k}{dt^k} y_{\mathrm{a}}(t) = \sum_{k=0}^{M} \beta_k \frac{d^k}{dt^k} x_{\mathrm{a}}(t)$$

with corresponding **system function** (Laplace transform of the impulse response)

$$H_{\mathrm{a}}(s) = \frac{\sum_{k=0}^{M} \beta_k s^k}{\sum_{k=0}^{N} \alpha_k s^k}.$$

Each combination of $N, M, \{\alpha_k\}, \{\beta_k\}$ corresponds to some arrangement of RLCs. Those implementation details are unimportant to us here.

The frequency response of a general analog filter is

$$H_{\mathrm{a}}(F) = H_{\mathrm{a}}(s)\Big|_{s=j2\pi F}.$$

Overview

- Design $N, M, \{\alpha_k\}, \{\beta_k\}$ using existing methods.
- Map from s plane to z-plane somehow to get a_k's and b_k's, *i.e.*, a rational system function corresponding to a discrete-time **linear constant coefficient difference equation**.

Can we achieve linear phase with an IIR filter? We should be able to use our analysis tools to answer this.

Recall that linear phase implies that

$$H(z) = z^{-N} H\left(z^{-1}\right),$$

so if z is a pole of the system function $H(z)$, then z^{-1} is also a pole. So any finite poles (*i.e.*, other than 0 or ∞) would lead to instability.

There are no causal stable IIR filters with linear phase.

Thus we design for the magnitude response, and see what phase response we get.

IIR filter design by bilinear transformation

Suppose we have used existing analog filter design methods to design an IIR analog filter with system function

$$H_{\rm a}(s) = \frac{\sum_{k=0}^{M} \beta_k s^k}{\sum_{k=0}^{N} \alpha_k s^k}.$$

For a sampling period $T_{\rm s}$, we now make the **bilinear transformation**

$$\boxed{s = \frac{2}{T_{\rm s}} \frac{1 - z^{-1}}{1 + z^{-1}}.}$$

This transformation can be motivated by the trapezoidal formula for numerical integration. See text for the derivation.

Define the discrete-time system function

$$\boxed{H(z) = H_{\rm a}(s)\Big|_{s = \frac{2}{T_{\rm s}} \frac{1 - z^{-1}}{1 + z^{-1}}}.}$$

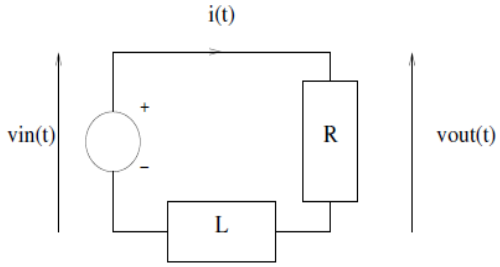
This transformation yields a **rational system function**, *i.e.*, a ratio of polynomials in z.

This $H(z)$ is a system function whose frequency response is related to the frequency response of the analog IIR filter.

Example. Consider a 1st-order analog filter with a single pole at $s = -\alpha$ ***Picture*** where $\alpha > 0$, with system function

$$H_{\rm a}(s) = \frac{1}{\alpha + s}.$$

How would you build this? Using the following RL "voltage divider," where $V_{\rm out}(s) = \frac{R}{R+sL} V_{\rm in}(s) \Longrightarrow H_{\rm a}(s) = \frac{R/L}{R/L+s}$.



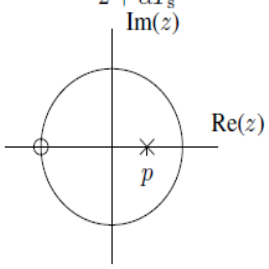
Applying the bilinear transformation to the above (Laplace domain) system function yields:

$$H(z) = H_{\rm a}(s)\Big|_{s = \frac{2}{T_{\rm s}} \frac{1 - z^{-1}}{1 + z^{-1}}} = \frac{1}{\alpha + \frac{2}{T_{\rm s}} \frac{1 - z^{-1}}{1 + z^{-1}}} = \frac{1 + z^{-1}}{\alpha(1 + z^{-1}) + \frac{2}{T_{\rm s}}(1 - z^{-1})} = \frac{1 + z^{-1}}{(\alpha + 2/T_{\rm s}) + (\alpha - 2/T_{\rm s}) z^{-1}}$$

$$= \frac{1}{\alpha + 2/T_{\rm s}} \frac{1 + z^{-1}}{1 - \frac{2/T_{\rm s} - \alpha}{2/T_{\rm s} + \alpha} z^{-1}} = \frac{1}{\alpha + 2/T_{\rm s}} \frac{1 + z^{-1}}{1 - p z^{-1}} = \frac{1}{\alpha + 2/T_{\rm s}} \left[\frac{-1}{p} + \frac{1 + 1/p}{1 - p z^{-1}} \right],$$

where $p = \dfrac{2 - \alpha T_{\rm s}}{2 + \alpha T_{\rm s}} \in (-1, 1)$.

What is $h[n]$? $h[n] = \frac{1}{\alpha + 2/T_{\rm s}} \left(-\frac{1}{p} \, \delta[n] + (1 + 1/p) p^n \, u[n] \right)$.



Where did zero at $z = -1$ come from?

Solving bilinear transformation for z in terms of s yields: $\boxed{z = \frac{2 + sT_s}{2 - sT_s}}$, so zero at $s = \infty$ maps to $z = -1$.

Note also that pole at $s = -\alpha$ maps to $z = p$.

- In general, the (finite) poles and zeros follow the mapping.
- Poles (or zeros) at $s = \infty$ map to $z = -1$.
- Real poles and zeros remain real, complex-conjugate pairs remain pairs.

What type of filters are $H_a(s)$ and $H(z)$? Both are poor lowpass filters.
This illustrates the method, but not the power! The utility is for more sophisticated analog IIR designs.

Picture : more s-domain poles and corresponding z-domain poles

The bilinear transformation

More generally, if the system function $H_a(s)$ is rational, *i.e.*, $H_a(s) = g\frac{\prod_i (s-s_i)}{\prod_j (s-p_j)}$, then for each term of the form $s-p$ we have

$$s - p \mapsto \frac{2}{T_s}\frac{1-z^{-1}}{1+z^{-1}} - p = \ldots = \frac{2}{T_s}\left(1 - \frac{pT_s}{2}\right)\frac{z - \frac{2+pT_s}{2-pT_s}}{z+1},$$

so we will have one root at $z = \frac{2+pT_s}{2-pT_s}$ and another one at $z = -1$.

$$z = \frac{2+sT_s}{2-sT_s}, \qquad s = \frac{2}{T_s}\frac{1-z^{-1}}{1+z^{-1}}.$$

See plot on next page.

- $\text{real}(s) > 0$ maps to z outside unit circle
- $\text{real}(s) < 0$ maps to z inside unit circle
- $s = \jmath 2\pi F$ maps to z on unit circle

$$z = \frac{2+\jmath 2\pi F T_s}{2-\jmath 2\pi F T_s} = \frac{r\,e^{\jmath\phi}}{r\,e^{-\jmath\phi}} = e^{\jmath 2\phi},$$

where $r = |2+\jmath 2\pi F T_s|$ and $\phi = \angle(2+\jmath 2\pi F T_s) = \arctan\left(\frac{2\pi F T_s}{2}\right)$.

Thus $z = e^{\jmath\omega}$, where $\omega = 2\phi$, so $\boxed{\omega = 2\tan^{-1}\left(\frac{2\pi F T_s}{2}\right).}$

- Conversely, if $z = e^{\jmath\omega}$ then

$$s = \frac{2}{T_s}\frac{1-e^{-\jmath\omega}}{1+e^{-\jmath\omega}} = \frac{2}{T_s}\frac{e^{\jmath\omega/2} - e^{-\jmath\omega/2}}{e^{\jmath\omega/2} + e^{-\jmath\omega/2}} = \frac{2}{T_s}\frac{\jmath\sin\omega/2}{\cos\omega/2} = \jmath\frac{2}{T_s}\tan\left(\frac{\omega}{2}\right) \Longrightarrow \boxed{F = \frac{2}{2\pi T_s}\tan\left(\frac{\omega}{2}\right).}$$

- This is a *nonlinear* relationship between the analog frequency and the digital frequency. It is called **frequency warping**. See plot on next page.

Because of this nonlinearity, a typical design procedure goes as follows.

- Determine desired frequency response in terms of analog frequencies, *e.g.*, Hz.
- Convert the desired frequencies into digital frequencies using $\omega = 2\pi F/F_s$, yielding ω_p, ω_s, etc.
- Map those digital frequencies into analog frequencies $F = \frac{2}{2\pi T_s}\tan(\omega/2)$.
  We can use any convenient T_s for this, *e.g.*, $T_s = 1$.
- Design an analog filter for those frequencies.
- Transform analog filter into digital filter using the bilinear transformation with that same T_s.

This is all built into commands such as MATLAB's `cheby1` routine.
Use MATLAB's `filtdemo` to experiment with various filter types.

The Chebyshev Type I filters are all-pole analog filters with equiripple behavior in the passband and monotonic in the stopband.

$$|H_a(F)|^2 = \frac{1}{1+\varepsilon^2 T_N^2(F/F_{\text{pass}})} \quad \text{where} \quad T_N(x) \triangleq \begin{cases} \cos\left(N\cos^{-1}x\right), & |x| \le 1 \\ \cosh(N\cosh^{-1}x), & |x| > 1 \end{cases}$$

is the Nth order **Chebyshev polynomial**.
There is **ripple** in the passband of amplitude $1/(1+\varepsilon^2)$ that is user-controlled.

Advantages of using digital filters

The following list gives some of the main advantages of digital over analog filters.

1. A digital filter is *programmable*, i.e. its operation is determined by a program stored in the processor's memory. This means the digital filter can easily be changed without affecting the circuitry (hardware). An analog filter can only be changed by redesigning the filter circuit.

2. Digital filters are easily *designed*, *tested* and *implemented* on a general-purpose computer or workstation.

3. The characteristics of analog filter circuits (particularly those containing active components) are subject to drift and are dependent on temperature. Digital filters do not suffer from these problems, and so are extremely *stable* with respect both to time and temperature.

4. Unlike their analog counterparts, digital filters can handle *low frequency* signals accurately. As the speed of DSP technology continues to increase, digital filters are being applied to high frequency signals in the RF (radio frequency) domain, which in the past was the exclusive preserve of analog technology.

5. Digital filters are very much more *versatile* in their ability to process signals in a variety of ways; this includes the ability of some types of digital filter to adapt to changes in the characteristics of the signal.

6. Fast DSP processors can handle complex combinations of filters in parallel or cascade (series), making the hardware requirements relatively *simple* and *compact* in comparison with the equivalent analog circuitry.

Operation of digital filters

In this section, we will develop the basic theory of the operation of digital filters. This is essential to an understanding of how digital filters are designed and used.

Suppose the "raw" signal which is to be digitally filtered is in the form of a voltage waveform described by the function

$$V = x(t)$$

where t is time.

This signal is sampled at time intervals h (the sampling interval). The sampled value at time $t = ih$ is

$$x_i = x(ih)$$

Thus the digital values transferred from the ADC to the processor can be represented by the sequence

$$x_0, x_1, x_2, x_3, \ldots$$

corresponding to the values of the signal waveform at

$$t = 0, h, 2h, 3h, \ldots$$

and $t = 0$ is the instant at which sampling begins.

At time $t = nh$ (where n is some positive integer), the values available to the processor, stored in memory, are

$$x_0, x_1, x_2, x_3, \ldots x_n$$

Note that the sampled values x_{n+1}, x_{n+2} etc. are not available, as they haven't happened yet!

The digital output from the processor to the DAC consists of the sequence of values

$$y_0, y_1, y_2, y_3, \ldots y_n$$

In general, the value of y_n is calculated from the values $x_0, x_1, x_2, x_3, .. x_n$. The way in which the y's are calculated from the x's determines the filtering action of the digital filter.

The transfer function can also be written in terms of the difference equation as

$$H(z) = \frac{b_0 + b_1\, z^{-1} + b_2\, z^{-2} + \cdots + b_M\, z^{-M}}{1 + a_1\, z^{-1} + a_2\, z^{-2} + \cdots + a_N\, z^{-N}} \tag{5}$$

or

$$H(z) = \frac{B(z)}{A(z)} \tag{6}$$

where

$$B(z) = \sum_{n=0}^{M} b(n)\, z^{-n} \tag{7}$$

$$A(z) = 1 + \sum_{n=1}^{N} a(n)\, z^{-n}. \tag{8}$$

$H(z)$ can also be written as

$$H(z) = \frac{z^{-M}}{z^{-N}} \cdot \frac{b_0\, z^M + b_1\, z^{M-1} + b_2\, z^{M-2} + \cdots + b_M}{z^N + a_1\, z^{N-1} + a_2\, z^{N-2} + \cdots + a_N}. \tag{9}$$

The the zeros of $H(z)$ are the roots of the polynomial

$$b_0\, z^M + b_1\, z^{M-1} + b_2\, z^{M-2} + \cdots + b_M \tag{10}$$

and the poles of $H(z)$ are the roots of the polynomial

$$z^N + a_1\, z^{N-1} + a_2\, z^{N-2} + \cdots + a_N. \tag{11}$$

In addition, if $N > M$, then $z = 0$ is a zero of multiplicity $N - M$; and if $M > N$, then $z = 0$ is a pole of multiplicity $M - N$.

A recursive IIR digital filter is an LTI system based on a difference equation of the form:

$$y(n) = -\sum_{k=1}^{N} a(k)\, y(n-k) + \sum_{k=0}^{M} b(k)\, x(n-k) \tag{1}$$

1. The impulse response $h(n)$ is infinite in length.

2. A system described by this type of difference equation is called an IIR (Infinite Impulse Response) filter, a recursive filter, or an autoregressive moving-average (ARMA) filter.

The output $y(n)$ of the filter can be written as

$$y(n) = \sum_{k=0}^{\infty} h(k)\, x(n-k). \tag{2}$$

As the impulse response is infinite, the convolutional sum is an infinite sum. The transfer function of the system $H(z)$ can be written in term of the impulse response as

$$H(z) = \sum_{k=0}^{\infty} h(k)\, z^{-k}. \tag{3}$$

The frequency response is therefore given by

$$H(e^{j\omega}) = \sum_{n=0}^{\infty} h(n)\, e^{-j\omega}. \tag{4}$$

The transfer function can also be written in terms of the difference equation as

$$H(z) = \frac{b_0 + b_1\, z^{-1} + b_2\, z^{-2} + \cdots + b_M\, z^{-M}}{1 + a_1\, z^{-1} + a_2\, z^{-2} + \cdots + a_N\, z^{-N}} \tag{5}$$

or

$$H(z) = \frac{B(z)}{A(z)} \tag{6}$$

where

$$B(z) = \sum_{n=0}^{M} b(n)\, z^{-n} \tag{7}$$

$$A(z) = 1 + \sum_{n=1}^{N} a(n)\, z^{-n}. \tag{8}$$

$H(z)$ can also be written as

$$H(z) = \frac{z^{-M}}{z^{-N}} \cdot \frac{b_0\, z^M + b_1\, z^{M-1} + b_2\, z^{M-2} + \cdots + b_M}{z^N + a_1\, z^{N-1} + a_2\, z^{N-2} + \cdots + a_N}. \tag{9}$$

The the zeros of $H(z)$ are the roots of the polynomial

$$b_0\, z^M + b_1\, z^{M-1} + b_2\, z^{M-2} + \cdots + b_M \tag{10}$$

and the poles of $H(z)$ are the roots of the polynomial

$$z^N + a_1\, z^{N-1} + a_2\, z^{N-2} + \cdots + a_N. \tag{11}$$

In addition, if $N > M$, then $z = 0$ is a zero of multiplicity $N - M$; and if $M > N$, then $z = 0$ is a pole of multiplicity $M - N$.

Recursive and non-recursive filters

For all the examples of digital filters discussed so far, the current output (*yn*) is calculated solely from the current and previous input values (*xn, xn-1, xn-2, ...*). This type of filter is said to be *non-recursive*.

A *recursive* filter is one which in addition to input values also uses previous *output* values. These, like the previous input values, are stored in the processor's memory.

The word *recursive* literally means "running back", and refers to the fact that previously-calculated output values go back into the calculation of the latest output. The expression for a recursive filter therefore contains

not only terms involving the input values (x_n, *xn-1, xn-2, ...*) but also terms in *yn-1, yn-2, ...*

From this explanation, it might seem as though recursive filters require more calculations to be performed, since there are previous output terms in the filter expression as well as input terms. In fact, the reverse is usually the case: to achieve a given frequency response characteristic using a recursive filter generally requires a much lower order filter (and therefore fewer terms to be evaluated by the processor) than the equivalent non-recursive filter.

Note

Some people prefer an alternative terminology in which a non-recursive filter is known as an FIR (or Finite Impulse Response) filter, and a recursive filter as an IIR (or Infinite Impulse Response) filter.

These terms refer to the differing "impulse responses" of the two types of filter. The impulse response of a digital filter is the output sequence from the filter when a *unit impulse* is applied at its input. (A unit impulse is a very simple input sequence consisting of a single value of 1 at time *t = 0*, followed by zeros at all subsequent sampling instants).

An FIR filter is one whose impulse response is of finite duration. An IIR filter is one whose impulse response theoretically continues for ever because the recursive (previous output) terms feed back energy into the filter input and keep it going. The term IIR is not very accurate

because the actual impulse responses of nearly all IIR filters reduce virtually to zero in a finite time. Nevertheless, these two terms are widely used.

Example of a recursive filter

A simple example of a recursive digital filter is given by

$$y_n = x_n + y_{n-1}$$

In other words, this filter determines the current output (*y*n) by adding the current input (*x*n) to the previous output (*y*n-1):

$$\begin{aligned} y_0 &= x_0 + y_{-1} \\ y_1 &= x_1 + y_0 \\ y_2 &= x_2 + y_1 \\ y_3 &= x_3 + y_2 \\ &\ldots \text{ etc} \end{aligned}$$

Note that *y*-1 (like *x*-1) is undefined, and is usually taken to be zero.

Let us consider the effect of this filter in more detail. If in each of the above expressions we substitute for *yn-1* the value given by the previous expression, we get the following:

$$\begin{aligned} y_0 &= x_0 + y_{-1} = x_0 \\ y_1 &= x_1 + y_0 = x_1 + x_0 \\ y_2 &= x_2 + y_1 = x_2 + x_1 + x_0 \\ y_3 &= x_3 + y_2 = x_3 + x_2 + x_1 + x_0 \\ &\text{etc} \end{aligned}$$

Thus we can see that *y*n, the output at *t* = *nh*, is equal to the sum of the current input *x*n and all the previous inputs. This filter therefore sums or *integrates* the input values, and so has a similar effect to an analog integrator circuit.

This example demonstrates an important and useful feature of recursive filters: the economy with which the

output values are calculated, as compared with the equivalent non-recursive filter. In this example, each output

is determined simply by adding two numbers together. For instance, to calculate the output at time *t* = 10*h*, the

recursive filter uses the expression

$$y_{10} = x_{10} + y_9$$

To achieve the same effect with a non-recursive filter (i.e. without using previous output values stored in

memory) would entail using the expression

$$y_{10} = x_{10} + x_9 + x_8 + x_7 + x_6 + x_5 + x_4 + x_3 + x_2 + x_1 + x_0$$

This would necessitate many more addition operations as well as the storage of many more values in memory.

Order of a recursive (IIR) digital filter

The *order* of a digital filter was defined earlier as the number of previous inputs which have to be stored in order to generate a given output. This definition is appropriate for non-recursive (FIR) filters, which use only

the current and previous inputs to compute the current output. In the case of recursive filters, the definition can be extended as follows:

The order of a recursive filter is the largest number of previous input *or* output values required to compute the current output.

This definition can be regarded as being quite general: it applies both to FIR and IIR filters.

For example, the recursive filter discussed above, given by the expression

$$y_n = x_n + y_{n-1}$$

is classed as being of first order, because it uses one previous output value (*yn-1*), even though no previous inputs are required.

In practice, recursive filters usually require the *same number of previous inputs and outputs*. Thus, a first-order recursive filter generally requires one previous input (x_{n-1}) and one previous output (y_{n-1}), while a second-order recursive filter makes use of two previous inputs (x_{n-1} and x_{n-2}) and two previous outputs (y_{n-1} and y_{n-2}); and so on, for higher orders.

Note that a recursive (IIR) filter must, by definition, be of at least first order; a zero-order recursive filter is an impossibility.

Coefficients of recursive (IIR) digital filters

From the above discussion, we can see that a recursive filter is basically like a non-recursive filter, with the addition of extra terms involving previous inputs (y_{n-1}, y_{n-2} etc.).

A first-order recursive filter can be written in the general form

$$y_n = \frac{(a_0 x_n + a_1 x_{n-1} - b_1 y_{n-1})}{b_0}$$

Note the minus sign in front of the "recursive" term $b_1 y_{n-1}$, and the factor *(1/b0)* applied to all the coefficients. The reason for expressing the filter in this way is that it allows us to rewrite the expression in the following symmetrical form:

$$b_0 y_n + b_1 y_{n-1} = a_0 x_n + a_1 x_{n-1}$$

In the case of a second-order filter, the general form is

$$y_n = \frac{a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2} - b_1 y_{n-1} - b_2 y_{n-2}}{b_0}$$

The alternative "symmetrical" form of this expression is

$$b_0 y_n + b_1 y_{n-1} + b_2 y_{n-2} = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2}$$

Note the convention that the coefficients of the inputs (the *x*'s) are denoted by *a*'s, while the coefficients of the outputs (the *y*'s) are denoted by *b*'s.

How to Design IIR Filter

The typical procedure to design IIR filter is:

1. Specify filter specification.
2. Specify low pass analog filter prototype, and the available prototypes supported in Origin include Butterworth, Chebyshev Type I, Chebyshev Type II, and Elliptic.

Method	Squared Magnitude Response Function	Analog Filter Transfer Function

Butterworth	$\lvert H_a(j\Omega)\rvert^2 = \frac{1}{1+\Omega^{2N}}$	$H_a(s) = \frac{q(s)}{p(s)} = \frac{g}{(s-p_1)(s-p_2)\cdots(s$
Chebyshev Type I	$\lvert H_a(j\Omega)\rvert^2 = \frac{1}{1+\varepsilon^2 T_N^2(\Omega)}$	$H_a(s) = \frac{q(s)}{p(s)} = \frac{g}{(s-p_1)(s-p_2)\cdots(s$
Chebyshev Type II	$\lvert H_a(j\Omega)\rvert^2 = \frac{1}{1+(\varepsilon^2 T_N^2(\frac{1}{\Omega}))}$	$H_a(s) = \frac{q(s)}{p(s)} = g\frac{(s-q_1)(s-q_2)\cdots(s}{(s-p_1)(s-p_2)\cdots(s}$
Elliptic	$\lvert H_a(j\Omega)\rvert^2 = \frac{1}{1+\varepsilon^2 U_N^2(\Omega)}$	$H_a(s) = \frac{q(s)}{p(s)} = g\frac{(s-q_1)(s-q_2)\cdots(s}{(s-p_1)(s-p_2)\cdots(s}$

In the table above, Ω is the frequency, N is the filter order, ε is the maximum oscillation in the passband frequency response, T_N is the Chebyshev polynomial, U_N is the Jacobian elliptic function, g is the scalar gain, s is the plane of Laplace transform, q_k or q_i is the zero, and p_k or p_j is the pole.

3. Frequency transform for analog filter

Transform the low pass filter into a high pass, band pass, or band stop filter with desired cutoff frequency. In Origin, the state-space form will be used in the frequency transform calculation. Assume the original transfer function of the low pass filter is $H(s')$, and the transfer function after transform is $H(s)$.

- Low pass to low pass, which transforms an analog low pass filter with cutoff frequency of 1 rad/s into a low pass filter with any specified cutoff frequency.

$$H(s) = H(s')|_{s'=s/\omega_0}$$

- Low pass to high pass

$$H(s) = H(s')|_{s'=\omega_0/s}$$

- Low pass to band pass

$$H(s) = H(s')|_{s'=\frac{\omega_0}{B_\omega}\frac{(s/\omega_0)^2+1}{s/\omega_0}}$$

- Low pass to band stop

$$H(s) = H(s')|_{s'=\frac{B_\omega}{\omega_0}\frac{s/\omega_0}{(s/\omega_0)^2+1}}$$

where $\omega_0 = sqrt(\omega_1 * \omega_2)$ is the center frequency, $B_\omega = \omega_2 - \omega_1$ is the bandwidth, ω_1 and ω_2 are the lower and upper band edges respectively.

4. Convert analog filter into a digital filter.

To convert analog filter into a digital filter, Origin uses the bilinear transformation, which is defined by expression:

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

IIR FILTER DESIGN

A causal IIR filter implemented with a rational transfer function can not have linear-phase. Two general approaches are:

1. Ignore the phase response and design a filter so that $|H(e^{j\omega})|$ matches a desired function $D(\omega)$.
2. If the phase response is important, then the design problem becomes more complicated.

The most common method for designing standard IIR digital filters is to convert a classical analog filter in a digital one.

- Highly developed methods for the design of classical analog filters can be used for the design of digital filters. Formulas exist for the classic analog filters.
- This method is of limited flexibility because not all IIR digital filters can be obtained by converting classical analog filters. The conversion of an analog filter into a digital filter works very well for the design of standard types: low-pass, high-pass, band-pass, and band-reject filters. However, if one wants to place constraints on the filters, or design a non-standard type of response, then the classical analog filters can usually not be used.
- A problem with the classical analog filters is that there is no control over the phase of the frequency response. They are developed so as to have good magnitude response but the phase-response can be very nonlinear.

- When an analog filter is converted into a digital filter, you generally get an IIR digital filter (not an FIR one). That is why the conversion of an analog filter to a digital one is not used for FIR filter design.
- The conversion of an analog filter into a digital one generally yields a digital filter for which the numerator and denominator of $H(z)$ have the same degree. $M = N$.

IIR discrete time filter design by bilinear transformation

Introduction: Many design techniques for IIR discrete time filters have adopted ideas and terminology developed for analogue filters, and are implemented by transforming the transfer function of an analogue 'prototype' filter into the system function of a discrete time filter with similar characteristics. We therefore begin this section with a reminder about analogue filters.

Analogue filters: Classical theory for analogue filters operating below about 100 MHz is generally based on "lumped parameter" resistors, capacitors, inductors and operational amplifiers (with feedback) which obey LTI equations and differential equations: (i(t)=Cdv(t)/dt, v(t)=Ldi(t)/dt, v(t)=i(t)R, $v_0(t) = A\, v_i(t)$. Analysis of such LTI circuits gives a relationship between input x(t) and output y(t) in the form of a differential equation:

$$b_0 y(t) + b_1 \frac{dy(t)}{dt} + b_2 \frac{d^2 y(t)}{dt^2} + \cdots = a_0 x(t) + a_1 \frac{dx(t)}{dt} + a_2 \frac{d^2 x(t)}{dt^2} + \cdots$$

whose transfer function is of the form:

$$H_a(s) = \frac{a_0 + a_1 s + a_2 s^2 + \ldots + a_N s^N}{b_0 + b_1 s + b_2 s^2 + ,,, + b_M s^M}$$

This is a rational function of s of order MAX(N,M). Replacing s by $j\omega$ gives the frequency response

$H_a(j\omega)$, where ω denotes frequency in radians/second. For values of s with non-negative real parts,

$H_a(s)$ is the Laplace Transform of the analogue filter's impulse response $h_a(t)$. H(s) may be expressed in terms of its poles and zeros as:

$$H_a(s) = K \frac{(s - z_1)(s - z_2) \ldots (s - z_N)}{(s - p_1)(s - p_2) \ldots (s - p_M)}$$

There is a wide variety of techniques for deriving $H_a(s)$ to have a frequency response which approximates a specified function. For example, we have previously derived a general expression for the system function of an n^{th} order analogue Butterworth low-pass filter. Such analogue filters have an important property in common with IIR type discrete time filters: their impulse responses are also infinite.

"Impulse response invariant" technique:

The philosophy of this technique is to transform an analogue prototype filter into an IIR discrete time filter whose impulse response {h[n]} is a sampled version of the analogue filter's impulse response, multiplied by T. The impulse response invariant technique will be covered later.

Bilinear transformation technique:

This is the most common method for transforming the system function $H_a(s)$ of an analogue filter to the system function H(z) of an IIR discrete time filter. It is not the only possible transformation, but a very useful and reliable one.

Definition: Given analogue transfer function $H_a(s)$, replace s by

$$\frac{2}{T}\left[\frac{z-1}{z+1}\right]$$

to obtain H(z). For convenience we can take T=1.

Example : If $H_a(s) = 1 / (1 + RC\,s)$ then

$$H(z) = \frac{z+1}{(1+2RC)z + (1-2RC)} = K\,\frac{1+z^{-1}}{1+b_1 z^{-1}}$$

where $K = 1/(1+2RC)$ and $b_1 = (1-2RC)/(1+2RC)$

Properties:

(i) This transformation produces a function H(z) such that given any complex number z,
$H(z) = H_a(s)$ where $s = 2(z-1)/(z+1)$

(ii) The order of H(z) is equal to the order of $H_a(s)$

(iii) If $H_a(s)$ is causal and stable, then so is H(z).

(iv) $H(\exp(j\Omega)) = H_a(j\omega)$ where $\omega = 2\tan(\Omega/2)$

Proof of (iii): Let z_p be a pole of H(z).

Then s_p must be a pole of $H_a(s)$ where $s_p = 2(z_p - 1)/(z_p + 1)$.

Let $s_p = a + jb$. Then $a < 0$ as $H_a(s)$ is causal & stable.

Now $(z_p + 1)(a + jb) = 2(z_p - 1)$, therefore $z_p = (a + 2 + jb)/(-a + 2 - jb)$ and

$$|z_p|^2 = \frac{(a+2)^2 + b^2}{(2-a)^2 + b^2} < 1 \quad \text{if } a < 0$$

Hence if all poles of $H_a(s)$ have real parts less than zero, then all poles of H(z) must lie inside the unit circle.

Proof of (iv): When $z = \exp(j\Omega)$, then

$$\frac{\exp(j\Omega)-1}{} \qquad \frac{2(e^{j\Omega/2} - e^{-j\Omega/2})}{}$$

$$s = 2\frac{\qquad}{\qquad} = \frac{\qquad}{\qquad} = 2j\tan(\Omega/2)$$

$$\exp(j\Omega)+1 \qquad e^{j\Omega/2} + e^{-j\Omega/2}$$

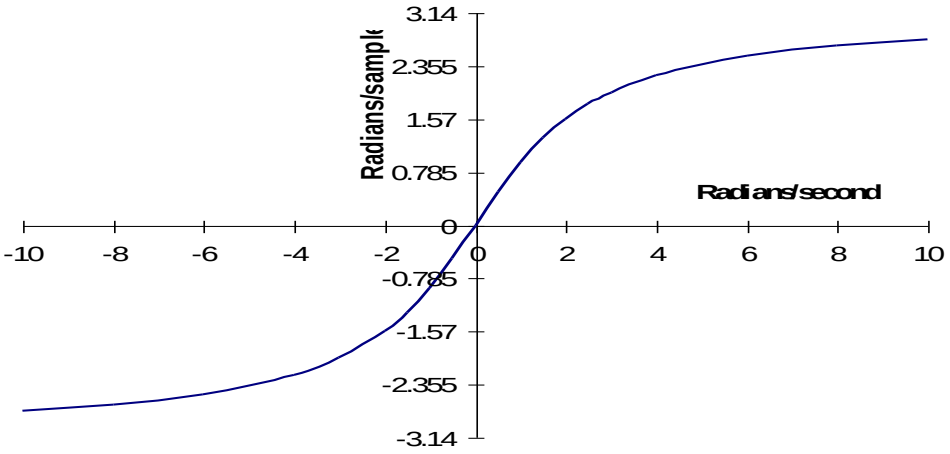


Fig 3.1: Frequency warping

Frequency warping: By property (iv) the discrete time filter's frequency response H(exp(jΩ)) at relative frequency Ω will be equal to the analogue frequency response $H_a(j\omega)$ with $\omega = 2\tan(\Omega/2)$. The graph of Ω against ω in fig 3.1, shows how ω in the range $-\infty$ to ∞ is mapped to Ω in the range $-\pi$ to π. The mapping is reasonably linear for ω in the range -2 to 2 (giving Ω in the range $-\pi/2$ to $\pi/2$), but as ω increases beyond this range, a given increase in ω produces smaller and smaller increases in Ω. Comparing the analogue gain response shown in fig 3.2(a) with the discrete time one in fig. 3.2(b) produced by the transformation, the latter becomes more and more compressed as $\Omega \rightarrow \pm\pi$. This "frequency warping" effect must be taken into account when determining a suitable $H_a(s)$ prior to the bilinear transformation.

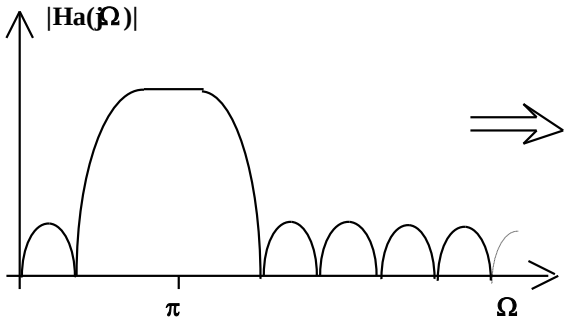


Fig 3.2(a): Analogue gain response

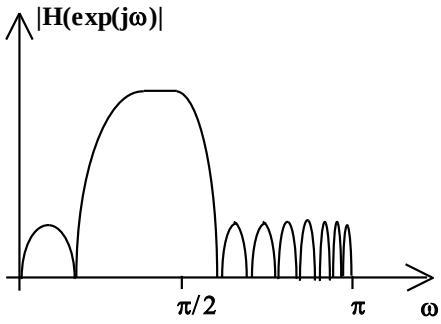


Fig 3.2(b): Effect of bilinear transformation

Design of an IIR low-pass filter by the bilinear transformation method:

Given the required cut-off frequency Ω_c in radians/sample:-

(i) Find $H_a(s)$ for an analogue low-pass filter with cut-off $\omega_c = 2\tan(\Omega_c/2)$ radians/sec. (ω_c is said to be the "pre-warped" cut-off frequency).

(ii) Replace s by 2(z - 1)/(z + 1) to obtain H(z).

(iii) Rearrange the expression for H(z) and realise by biquadratic sections.

Example 3.2 : Design a second order Butterworth-type IIR lowpass filter with $\Omega_c = \pi / 4$.

Solution: Pre-warped frequency $\omega_c = 2\tan(\pi / 8) = 0.828$

For an analogue Butterworth low-pass filter with cut-off frequency 1 radian/second:

$$H_a(s) = 1 / (1 + \sqrt{2}\, s + s^2)$$

Replace s by s / 0.828, then replace s by 2(z - 1)/(z + 1) to obtain:

$$H(z) = \frac{z^2 + 2z + 1}{10.3\, z^2 - 9.7\, z + 3.4} = 0.097 \left(\frac{1 + 2\, z^{-1} + z^{-2}}{1 - 0.94\, z^{-1} + 0.33\, z^{-2}} \right)$$

which may be realised by the signal flow graph in fig 3.3. Note the extra multiplier scaling the input by 0.097 .

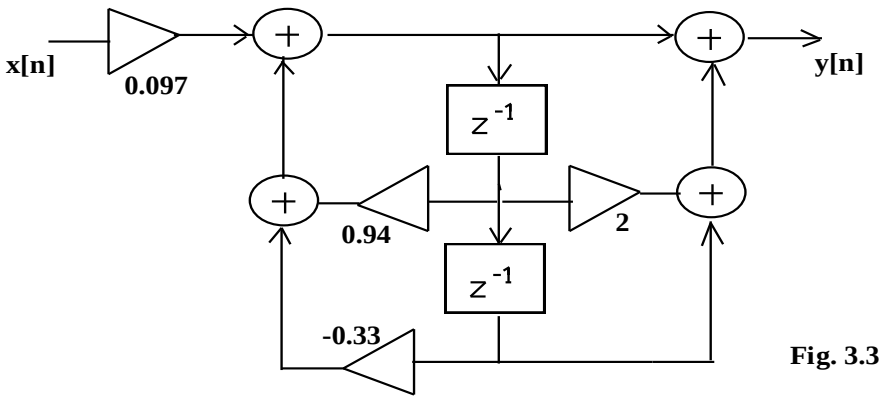


Fig. 3.3

Higher order IIR digital filters: Recursive filters of order greater than two are highly sensitive to quantisation error and overflow. It is normal, therefore, to design higher order IIR filters as cascades of bi-quadratic sections.

Example 3.3: A Butterworth-type IIR low-pass digital filter is needed with 3dB cut-off at one sixteenth of the sampling frequency f_s, and a stop-band attenuation of at least 24 dB for all frequencies above $f_s / 8$. (a) What order is needed? (b) Design it.

Solution:

(a) The relative cut-off frequency is $\pi/8$.

The pre-warped cut-off frequency: $\omega_c = 2 \tan((\pi/8)/2) \approx 0.40$ radians/second.

For an n^{th} order Butterworth low-pass filter with cutoff ω_c, the gain is:

$$|H_a(j\omega)| = \frac{1}{\sqrt{[1 + (\omega/0.4)^{2n}]}}$$

The gain of the IIR filter must be less than -24dB at the relative frequency $\omega = \pi/4$.

This means that the gain of the analogue prototype filter must be less than -24 dB at the pre-warped frequency corresponding to $\Omega = \pi/4$, i.e. at $\omega = 2 \tan(\pi/8) \approx 0.83$

Therefore, $20 \log_{10} (1/\sqrt{[1+(.83/.4)^{2n}]}) < -24$

i.e., $\sqrt{[1 + (2.1)^{2n}]} > 10^{1.2}$

Hence n must be such that $1 + (2.1)^{2n} > 10^{2.4} = 252$.

We find that $n = 4$ is the smallest possible.

(b) Formula for 4th order Butterworth 1 radian/sec low-pass system function:

$$H_a(s) = \left(\frac{1}{1 + 0.77\,s + s^2} \right) \left(\frac{1}{1 + 1.85\,s + s^2} \right)$$

Scale the analogue cut-off frequency to ω_c by replacing s by s / 0.4.

Then replace s by 2 (z - 1)/(z +1) to obtain:

$$H(z) = 0.033 \left[\frac{1 + 2z^{-1} + z^{-2}}{1 - 1.6z^{-1} + .74z^{-2}} \right] 0.028 \left[\frac{1 + 2z^{-1} + z^{-2}}{1 - 1.365z^{-1} + 0.48z^{-2}} \right]$$

H(z) may be realised in the form of cascaded bi-quadratic sections as shown in fig 3.4.

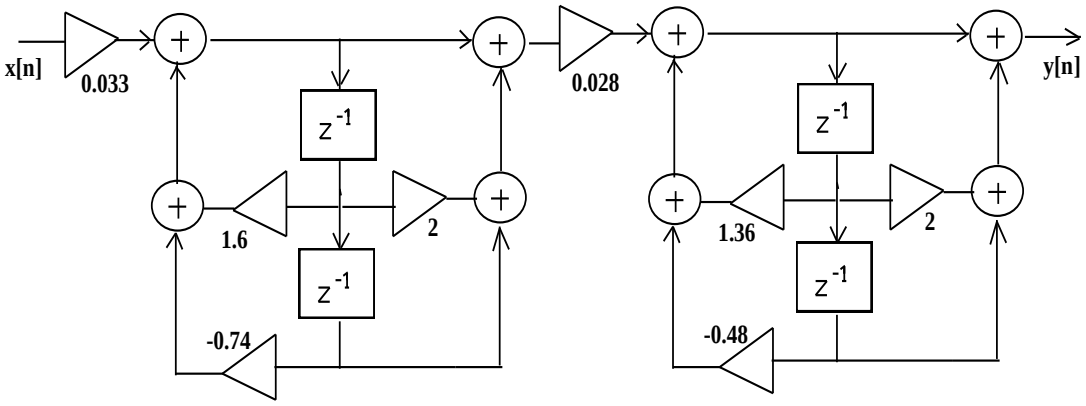


Fig. 3.4: Fourth order IIR Butterworth filter with cut-off fs/16

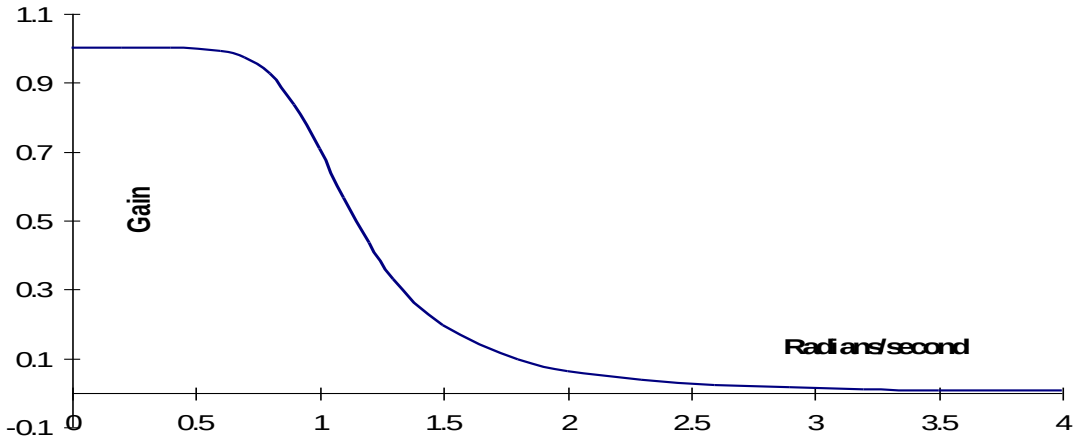


Fig. 3.5(a) Analogue 4th order Butterworth gain response

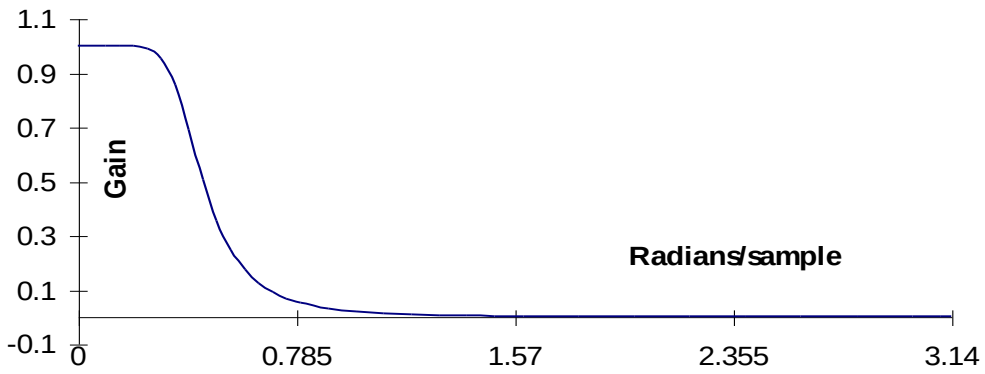


Fig. 3.5(b): Gain response of 4th order IIR filter

Fig. 3.5(a) shows the gain response for the 4th order Butterworth low-pass filter whose transfer function was used here as a prototype. Fig 3.5(b) shows the gain response of the derived digital filter which, like the analogue filter, is 1 at zero frequency and 0.707 at the cut-off frequency. Note however that the analogue gain approaches 0 as $\omega \to \infty$ whereas the gain of the digital filter becomes exactly zero at $\Omega = \pi$. The shape of the Butterworth gain response is "warped" by the bilinear transformation. However, the 3dB point occurs exactly at Ω_c for the digital filter, and the cut-off rate becomes sharper and sharper as $\Omega \to \pi$ because of the compression as $\omega \to \infty$.

IIR discrete time high-pass band-pass and band-stop filter design:

The bilinear transformation may be applied to analogue transfer functions obtained by means of the high-pass, band-pass and band-stop frequency transformations considered earlier. As in the low-pass case, cut-off frequencies must be pre-warped to find appropriate analogue cut-off frequencies. For band-pass and band-stop filters, there are two cut-off frequencies to be pre-warped.

Example : Design a 4th order band-pass filter with $\Omega_L = \pi / 4$, $\Omega_u = \pi / 2$.

Solution: Prewarp both cutoff frequencies:

$$\omega_L = 2 \tan ((\pi/4)/2) = 2 \tan(\pi/8) = 0.828 ,$$

$$\omega_u = 2 \tan((\pi/2)/2)) = 2 \tan(\pi/4) = 2$$

Now derive $H_a(s)$ for a 4th order analogue band-pass filter, with pass-band ω_L to ω_u, starting from a 2nd order Butterworth 1 radian/sec prototype:

$$H_a(s) = 1 / (s^2 + \sqrt{2}\, s + 1).$$

This requires s to be replaced by $(s^2 + 1.66) / 1.17\ s$ and produces an analogue system function whose denominator is a 4th order polynomial in s.

$$\frac{1.37\ s^2}{s^4 + 1.65\ s^3 + 4.69\ s^2 + 2.75\ s + 2.76}$$

It is now necessary to express the denominator as the product of two second order polynomials in s. This may be done by running a "root finding" computer program.

Such a program is "ROOTS87.EXE" (available on application). Running this program produces the following output:-

ENTER ORDER: 4

R(0): 2.76 R(1): 2.75 R(2): 4.69 R(3): 1.65 R(4): 1

ROOTS ARE:-

RE: -0.5423 IM: 1.7104 RE: -0.2827 IM: -0.8817

RE: -0.5423 IM: -1.7104 RE: -0.2827 IM: 0.8817

Therefore,

$$H_a(s) = \frac{1.37\ s^2}{(s - [-0.54+1.17j])(s - [-0.54-1.17j])(s - [-0.28+0.88j])(s - [-0.28-0.88j])}$$

Combining the first two factors and the last two factors of the denominator, which have complex conjugate roots, we obtain $H_a(s)$ factorised into second order sections:-

$$H_a(s) = \frac{1.37\ s^2}{(s^2 + 1.085\ s + 3.22)(s^2 + 0.565\ s + 0.857)}$$

Replacing s by 2(z - 1)/(z + 1) gives the transfer function:-

$$H(z) = \frac{5.48\,(z-1)^2\,(z+1)^2}{(9.4\,z^2 - 1.57\,z + 5)\,(6\,z^2 - 6.3\,z + 3.7)}$$

Rearranging into two bi-quadratic sections (we can do this in different ways) we obtain:

$$H(z) = 0.098 \left[\frac{1 - 2z^{-1} + z^{-2}}{1 - 0.167\,z^{-1} + 0.535\,z^{-2}}\right]\left[\frac{1 + 2z^{-1} + z^{-2}}{1 - 1.05\,z^{-1} + 0.62\,z^{-2}}\right]$$

whose gain response is shown in fig. 3.6.

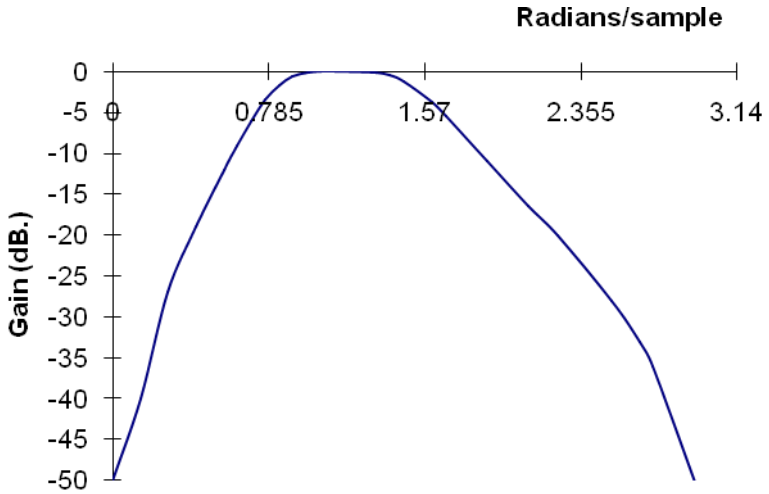


Fig. : Gain response of band-pass IIR filter

The design of analogue band-pass and band-stop system functions $H_a(s)$ as required for realisation as analogue or digital filters can be greatly simplified if the filters can be considered "wide-band", i.e. where $\omega_U >> 2\omega_L$ radians/second. In this case , it is reasonable to express $H_a(s) = H_{LP}(s)\,H_{HP}(s)$ where for a band-pass filter $H_{LP}(s)$ is the analogue system function of a low-pass filter cutting off at $\omega=\omega_U$ and $H_{HP}(s)$ is for a high-pass filter cutting off at $\omega=\omega_L$. $H_{LP}(s)$

and $H_{HP}(s)$ can now be designed separately and also transformed separately to functions of z via the bilinear (or other) transformation. Thus we obtain the transfer function $H(z) = H_{LP}(z)$ $H_{HP}(z)$ which may be realised as a serial cascade of two digital filters realising $H_{LP}(z)$ and $H_{HP}(z)$. Of course each of $H_{LP}(z)$ and $H_{HP}(z)$ may in itself be a cascade of several second or first order sections. This approach does not work very well for "narrow-band" filters where the analogue frequencies (i.e. the pre-warped frequencies if we are using the bilinear transformation) do not satisfy $\omega_U >> 2\omega_L$. In this case we have to use the frequency band transformation method as outlined above (which generally involves factorising fourth order polynomials by computer).

Digital Filter Design - Background

In this section, some background information is provided to clarify why the particular type of filter is chosen.

IIR Filters

One type of digital filter is the Infinite Impulse Response (IIR) filter, which is not as well supported and is generally used in the lower sample rates (i.e., < 200kHz). The IIR uses feedback in order to compute outputs, and it is known as a recursive filter.

Advantages of the IIR Filter:

1.Better magnitude response

2.Fewer coefficients

3.Less storage is required for storing variables

4.A lower delay

5. It is closer to analog models

A number of different classical IIR filters are available.

Butterworth

The Butterworth filter provides the best approximation to the ideal lowpass filter response at analog frequencies. Passband and stopband response is maximally flat.

Bessel

Analog Bessel lowpass filters have maximally flat group delay at zero frequency and retain nearly constant group delay across the entire passband. Filtered signals therefore maintain their waveshapes in the passband frequency range. Frequency mapped and digital Bessel filters, however, do not have this maximally flat property. Bessel filters generally require a higher filter order than other filters for satisfactory stopband attenuation.

IIR Filter Expressions

IIR Filters are recursive: the output is fed back to make a contribution. The expression for the IIR is shown below; note that a delayed version of the y(n) output plays a part in the output:

$$y(n) = \sum_{i=0}^{n} b(i)x(n-i) - \sum_{i=0}^{m} a(i)y(m-i)$$

a(i) and b(i) are the coefficients of the IIR filter. Another way to express a IIR Filter is as a transfer function with numerator coefficients “bi” and denominator coefficients “ai”:

$$H(z) = \frac{\sum_{i=0}^{n} b_i z^{-i}}{1 + \sum_{i=0}^{m} a_i z^{-i}}$$

3.3 IIR Filter Structures

Direct Form II

The Direct Form I architecture description noted that the forward and reverse FIR filter stages can be swapped, which creates a centre consisting of two columns of delay elements. From this, one column can be formed; hence, this type of structure is known as “canonical”, meaning it requires the minimum amount of storage.

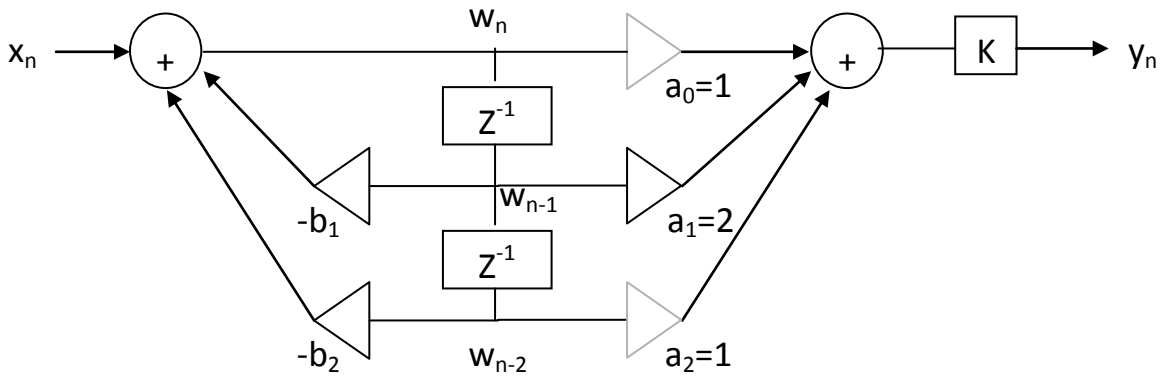


Figure 1: Direct Form II representation of a biquad

Biquad

The Biquad filter structure is that of a Direct Form-II, but it includes a second-order numerator and denominator coefficient (i.e., it is simply two poles and two zeros). This structure is used in FPGA/DSP implementations, because it is not terribly sensitive to quantization effects.

Butterworth Biquad:

The butterworth biquad expression is:

$$H(z) = \frac{1 + 2*z^{-1} + z^{-2}}{1 + b_1*z^{-1} + b_2*z^{-2}}$$

The coefficients in the nominator have the charm that simplify the calculations such that no multiplier is

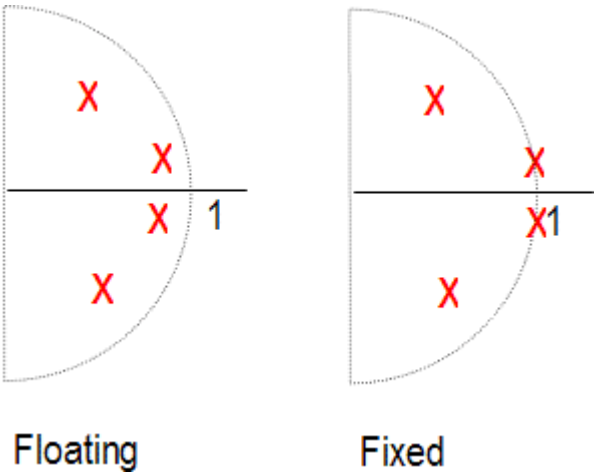
needed and the entire nominator can be calculated using shift and accumulators. Multiplications are expensive operations in FPGA/DSP implementations.

Fixed Point Implementations

Several issues must be examined in detail to ensure satisfactory fixed-point operation of the IIR filter:

1) Coefficient/Internal Quantization
2) Wraparound/Saturation
3) Scaling

Coefficient/Internal Quantization In order to examine the effect of quantization, it is useful to look at the pole/zero plot. This shows how the zeros (depths in the frequency response plot) and poles (peaks in the frequency response plot) are positioned. In fact, an issue with IIR stability relates to the denominator coefficients and their positions, as poles, on the pole/zero plot:



The poles for the floating-point version of the plot are shown on the left; they are within the unit circle (i.e., the values of the coefficients are less than 1). Once the coefficients are quantized, these poles move, which affects the frequency response. If they move onto the unit circle (i.e., the poles equal "1"), you potentially have an oscillator; If the poles become greater than 1, the filter becomes unstable.

Wraparound/Saturation

A fixed-point implementation has a certain bit width, and hence has a range. Calculations may cause the filter to exceed its maximum/minimum ranges. For example, let's consider a 2's complement value of ' 01111000'(+120) + '00001001'(+9) = '10000001' =(-127). The large positive number becomes a large negative number; this is known as "wraparound", and it can cause large errors.

Scaling

There are two methods of dealing with overflows:

1. If scaling is used, values can never overflow. DSP processors tend to use different kinds of scaling in order to fit within their fixed structure.

2. Use saturation logic. In our example, the results would be '01111111'(+127).

3.4 The Artifacts of IIR filters

The main artifacts of IIR filtering are the quantization noise and the limit cycle oscillations.

The truncation or rounding of the IIR accumulator at the output of filter creates quantization noise. This noise is fed into the filter recursive path. The noise is multiplied by the IIR recursive path transfer function. The impact of this noise source is very significant in the low cutoff frequency filters of the second order, since the recursive path gain is proportional to the second power of Fc/Fs ratio. The filter stages with high Q can also suffer from this effect because the gain is proportional to Q.

The best way to reduce the quantization noise is improve the arithmetic accuracy. For a multi-stage filter, the noise contributions of the stages can add.

The other way to reduce the quantization noise is the noise shaping. Noise shaping is feeding the accumulator truncation error back into the filter. That allows for better SNR at low frequencies for the cost of an increased noise at the high frequencies. The noise shaping with higher order error feedback can significantly improve the SNR, however the added complexity and limited performance makes it less attractive, then the increased precision arithmetic.

The limit cycles are the low amplitude oscillations which may occur in IIR filters. The reason for those oscillations is a finite precision of the arithmetic operations. The limit cycle existence depends on the particular filter coefficients and the input data. The filters with high Q, low Fc/Fs ratio and the rounding of the accumulator at the output rather then with truncation have a higher probability of a limit cycle behavior.

Usually the amplitude of limit cycle oscillations does not exceed several LSBs. The methods to avoid the limit cycles are the following:

- Improve the precision of filter arithmetic
- Implement a center clipping nonlinear function
- Dithering (adding a random noise with the amplitude of several LSBs)
- Gating: blocking the filter if the energy of the signal is below a certain limit

Implementation

Block Overview

The filter is implemented as part of the fsfb_calc block in order to share FPGA resources, particularly multipliers. Review fsfb_calc.doc first.

Block Functionality

Block Data Flow

Block Location and Block Interface within System

First Stage Feedback Filter Queue

This 64 x 32b RAM block stores the filtered output calculation results. The width of this queue is the same as the wishbone data. It is important to note that the filter results are not double buffered, since delay is acceptable in reading filter results. When a wishbone read request comes in, the read starts at the beginning of the next frame, in order to be aligned with the frame boundaries.

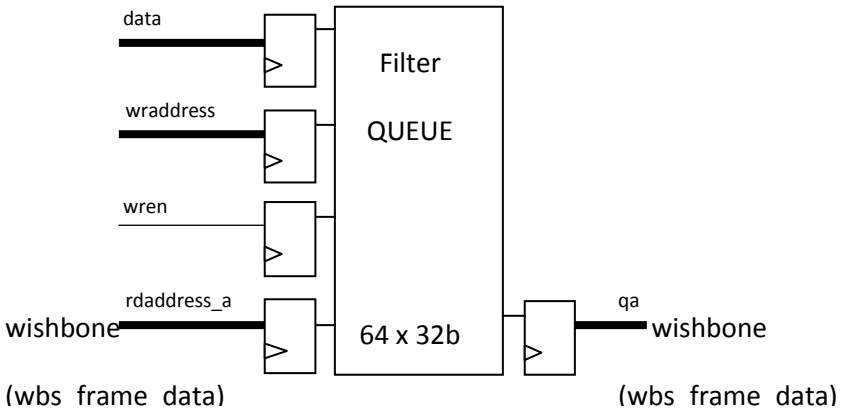


Figure 1: First Stage Feedback Filter Queue

First-Stage Feedback Filter Registers

The fsfb_filter_regs block instantiates 2 RAM blocks to store the previous 2 samples of wn, where wn is the interim filter calculation results. For details of the filter calculations, refer to the fsfb_calculations.doc where the implementation of the second-order Butterworth low-pass filter that is implemented.

The calculations are:

$$w_{temp} = b_1 * w_{n-1} + b_2 * w_{n-2}$$

$$w_n = x_n - w_{temp}/2^m$$

$$y_n = w_n + 2 * w_{n-1} + w_{n-2}$$

where x is the input to the filter and y is the output of the filter, b1 and b2 are the filter coefficients, m is the number of bits for the filter coefficients.

Note that the filter is reset through initialize_window_i signal. Each RAM block has 64 words and the word length is determined in the pack file by FLTR_DLY_WIDTH.

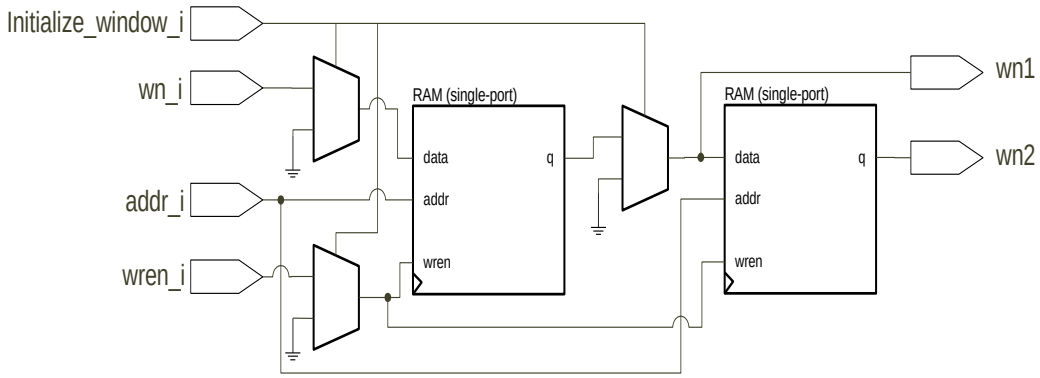


Figure 2: Filter Registers Storage

3.5 IIR-type digital filters

3.5.1. Introduction:

A general causal digital filter has the difference equation:

$$y[n] = \sum_{i=0}^{N} a_i x[n-i] - \sum_{k=1}^{M} b_k y[n-k]$$

which is of order max{ N,M }, and is recursive if any of the b_j coefficients are non-zero. A second order recursive digital filter therefore has the difference equation:

$$y[n] = a_0 x[n] + a_1 x[n-1] + a_2 x[n-2] - b_1 y[n-1] - b_2 y[n-2]$$

A digital filter with a recursive linear difference equation can have an infinite impulse-response. Remember that the frequency-response of a digital filter with impulse-response {h[n]} is:

$$H(e^{j\Omega}) = \sum_{n=-\infty}^{\infty} h[n] e^{-j\Omega n}$$

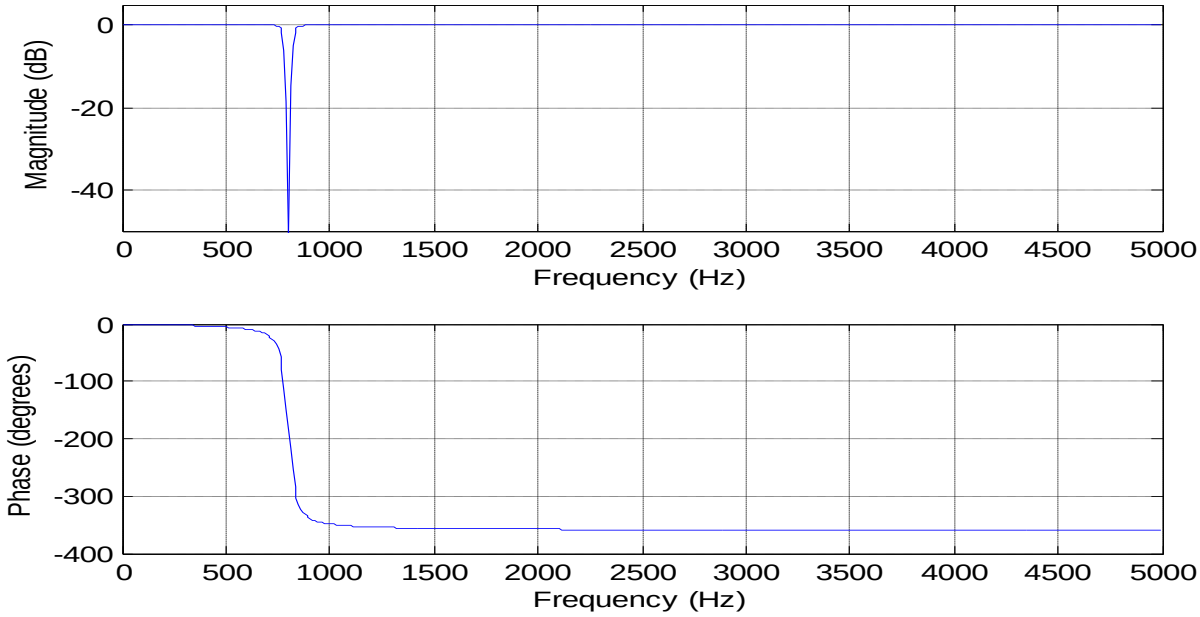
Design of a notch filter by MATLAB: Modified in 2009-10

Assume we wish to design a 4th order 'notch' digital filter to eliminate an unwanted sinusoid at 800 Hz without severely affecting rest of signal. The sampling rate is FS = 10 kHz.

One simple way is to use the MATLAB function 'butter' as follows:

```
FS=10000;
FL = 800 – 25 ;  FU = 800+25;
[a b] = butter(2, [FL FU]/(FS/2),'stop');
        a = [0.98  -3.43  4.96  -3.43  0.98]
        b= [  1    -3.47  4.96  -3.39  0.96]
freqz(a, b);
freqz(a, b, 512, FS);     % Better graph
axis([0 FS/2 -50 5]);     % Scales axes
```

The frequency-responses (gain and phase) produced by the final two MATLAB statements are as follows:



Since the Butterworth band-stop filter will have -3dB gain at the two cut-off frequencies

FL = 800-25 and FU=800+25, the notch has '-3 dB frequency bandwidth': 25 + 25 = 50 Hz.

Now consider how to implement the 4th order digital filter. The MATLAB function gave us:

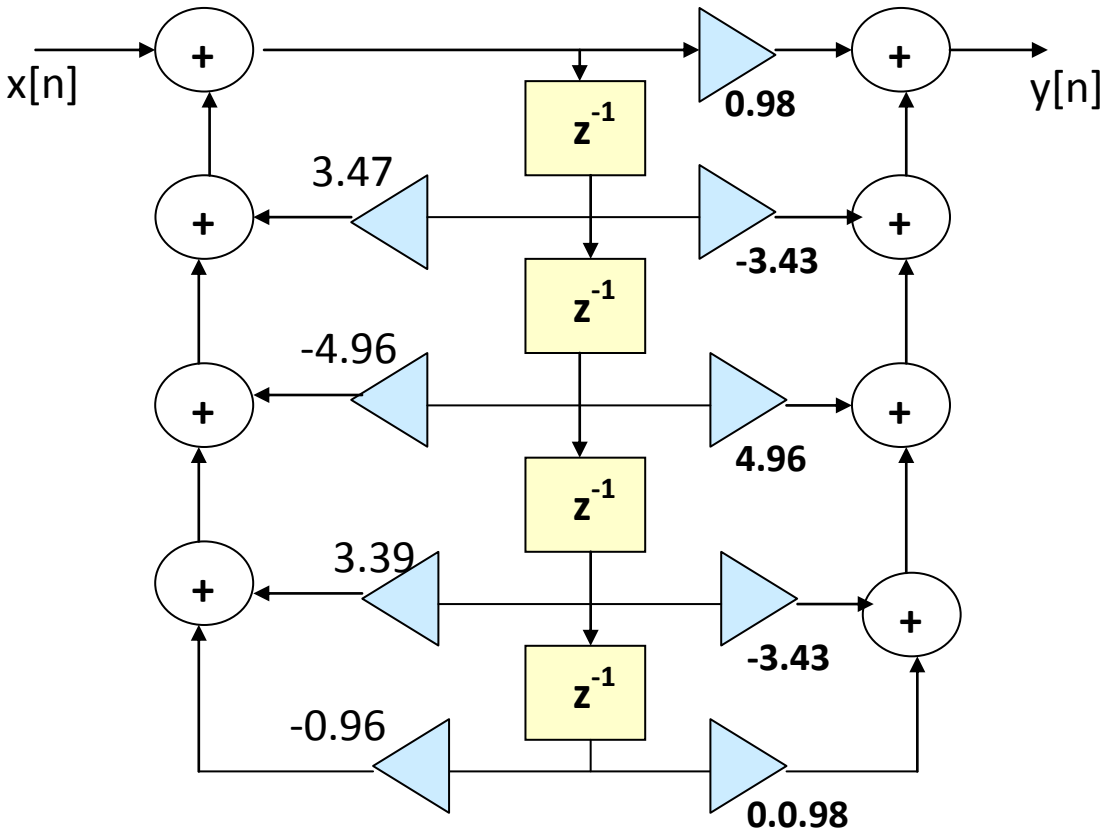
a = [0.98 -3.43 4.96 -3.43 0.98]

b= [1 -3.47 4.96 -3.39 0.96]

The transfer (System) Function is, therefore:

$$H(z)=\left(\frac{0.98-3.43z^{-1}+4.96z^{-2}-3.43z^{-3}+0.98z^{-4}}{1-3.47z^{-1}+4.96z^{-2}-3.39z^{-3}+0.96z^{-4}}\right)$$

A 'Direct Form II' implementation of the 4th order notch filter would have the signal-flow graph below:

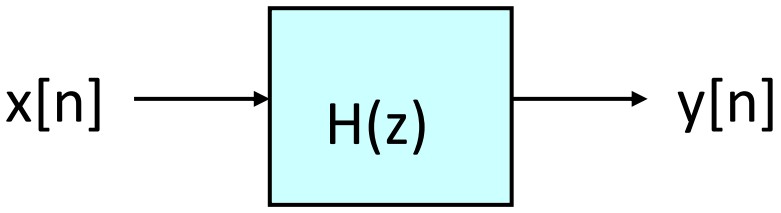


This implementation works fine in MATLAB. But 'direct form' IIR implementations of order greater than two are rarely used. Sensitivity to round-off error in coefficient values will be high.

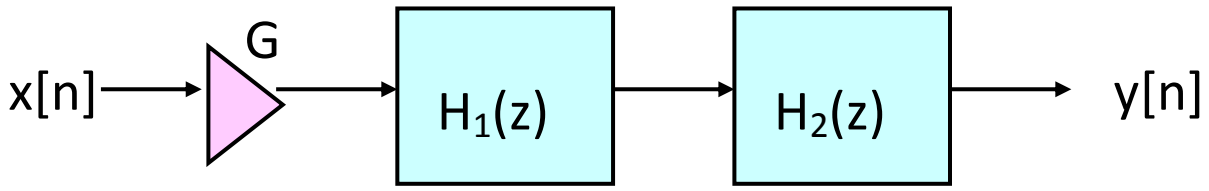
Also the range of 'intermediate' signals in the z^{-1} boxes will be high.

High word-length floating point arithmetic hides this problem, but in fixed point arithmetic, great difficulty occurs. Instead we use 'cascaded bi-quad sections'

Given a 4th order transfer function H(z). Instead of the direct form realization below:



we prefer to arrange two bi-quad sections, with a single leading multiplier G, as follows:



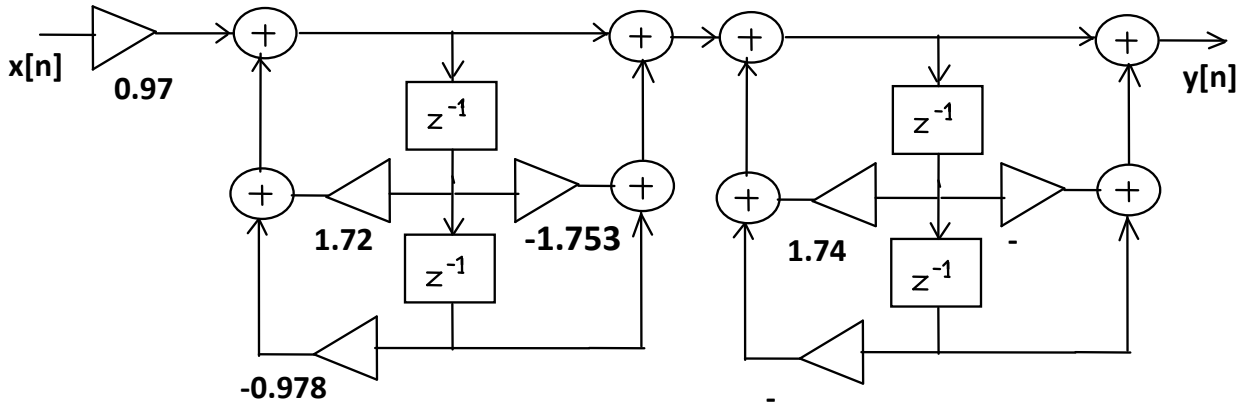
To convert the 4th order transfer function H(z) to this new form is definitely a job for MATLAB. Do it as follows after getting a & b for the 4th order transfer function, H(z), as before:

```
[a b] = butter(2, [FL FU]/(FS/2),'stop');

[SOS  G] = tf2sos(a,b)
```

- MATLAB responds with:

```
SOS = 1   -1.753   1   1   -1.722   0.9776

      1   -1.753   1   1   -1.744   0.9785

G = 0.978
```

In MATLAB, 'SOS' stands for 'second order section' (i.e. bi-quad) and the function 'tf2SOS' converts the coefficients in arrays 'a' and 'b' to the new set of coefficients stored in array 'SOS' and the constant G. The array SOS has two rows: one row for the first bi-quad section and one row for the second bi-quad section. In each row, the first three terms specify the non-recursive part and the second three terms specify the recursive part. Therefore H(z) may now be realized as follows:

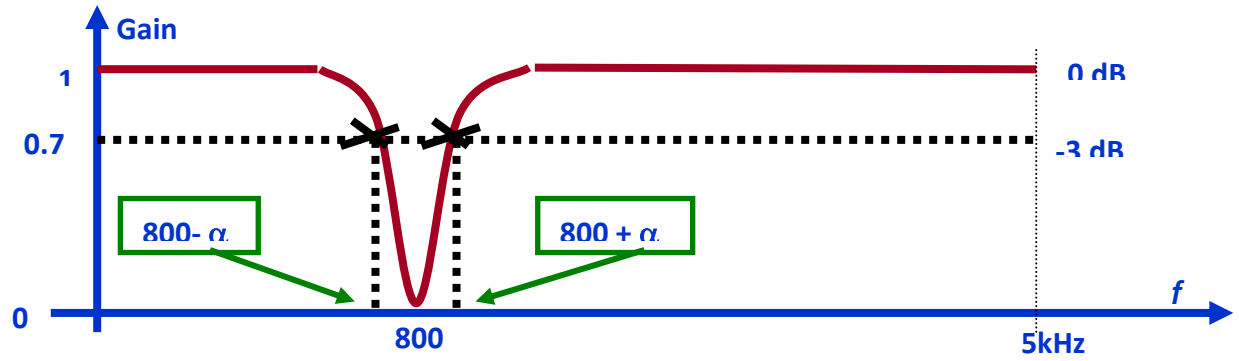


Fourth order IIR notch filter realised as two biquad (SOS) sections

This is now a practical and realizable IIR digital 'notch' filter, though we sometimes implement the single multiplier G =0.918 by two multipliers, one for each bi-quad section. More about this later.

Calculation of gain-response of notch filter:

How good is a notch filter? We can start to answer this question by specifying the filter's 3dB bandwidth i.e. the difference between the frequencies where the gain crosses 0.707 (-3dB). We should also ask what is the gain at the notch frequency (800 Hz in previous example); i.e. what is the 'depth' of the notch. If it is not deep enough either (i) increase the -3 dB bandwidth or (ii) increase the order. Do both if necessary. To 'sharpen' the notch, decrease the -3dB bandwidth, but this will make the notch less deep; so it may be necessary to increase the order to maintain a deep enough notch. This is an 'ad-hoc' approach – we can surely develop some theory later. It modifies the more formal approach, based on poles and zeroes, adopted last year.



Example: A digital filter with a sampling rate of 200 Hz is required to eliminate an unwanted 50 Hz sinusoidal component of an input signal without affecting the magnitudes of other components too severely. Design a 4th order "notch" filter for this purpose whose 3dB bandwidth is not greater than 3.2 Hz. (MATLAB calls this 2nd order.) How deep is the notch?

Solution:

```
FS=200;  FL=50-1.6; FU=50+1.6;
```

[a b]=butter(2,[FL,FU]/(FS/2), 'stop');

[SOS G] = tf2sos(a,b)

IIR digital filter design by bilinear transformation

Many design techniques for IIR discrete time filters have adopted ideas and terminology developed for analogue filters, and are implemented by transforming the system function, $H_a(s)$, of an analogue 'prototype' filter into the system function H(z) of a digital filter with similar, but not identical, characteristics.

For analogue filters, there is a wide variety of techniques for deriving $H_a(s)$ to have a specified type of gain-response. For example, it is possible to deriving $H_a(s)$ for an n^{th} order analogue Butterworth low-pass filter, with gain response:

$$G_a(\omega) = \frac{1}{\sqrt{1 + (\omega/\omega_C)^{2n}}}$$

It is then possible to transform $H_a(s)$ into H(z) for an equivalent digital filter. There are many ways of doing this, the most famous being the 'bilinear transformation'. It is not the only possible transformation, but a very useful and reliable one.

The bilinear transformation involves replacing s by (2/T) (z-1)/(z+1)], but fortunately, MATLAB takes care of all the detail and we can design a Butterworth low pass filter simply by executing the MATLAB statement:

[a b] = butter(N, *fc*)

N is the required order and *fc* is the required '3 dB' cut-off frequency normalised (as usual with MATLAB) to $f_S/2$. Analogue Butterworth filters have a gain which is zero in the pass-band and falls to -3 dB at the cut-off frequency. These two properties are preserved by the bilinear transformation, though the traditional Butterworth shape is changed. The shape change is caused by a process referred to as 'frequency warping'. Although the gain-response of the digital filter is consequently rather different from that of the analogue Butterworth gain response it is derived from, the term 'Butterworth filter' is still applied to the digital filter. The order of H(z) is equal to the order of $H_a(s)$

Frequency warping:

It may be shown that the new gain-response $G(\Omega) = G_a(\omega)$ where $\omega = 2\tan(\Omega/2)$**.** The graph of Ω against ω below, shows how ω in the range $-\infty$ to ∞ is mapped to Ω in the range $-\pi$ to π. The

mapping is reasonably linear for ω in the range -2 to 2 (giving Ω in the range $-\pi/2$ to $\pi/2$), but as ω increases beyond this range, a given increase in ω produces smaller and smaller increases in Ω. The effect of frequency warping is well illustrated by considering the analogue gain-response shown in fig 5.17(a). If this were transformed to the digital filter gain response shown in fig 5.17(b), the latter would become more and more compressed as $\Omega \rightarrow \pm \pi$.

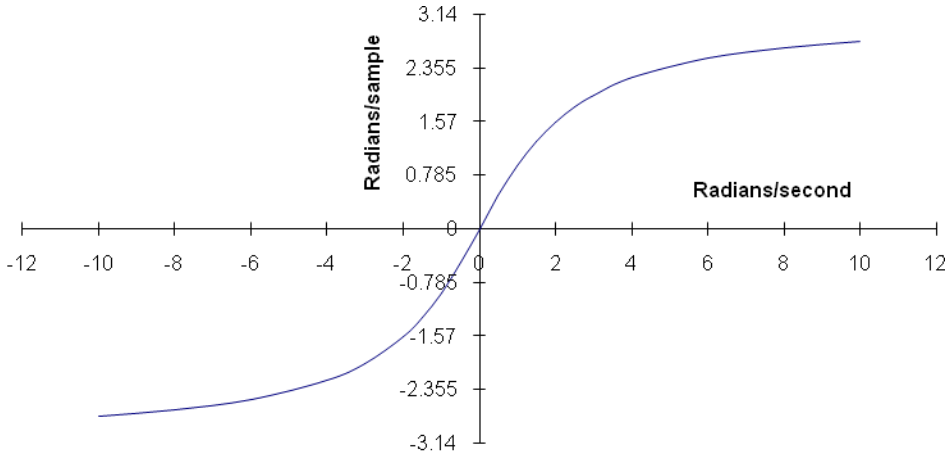


Fig : Frequency Warping

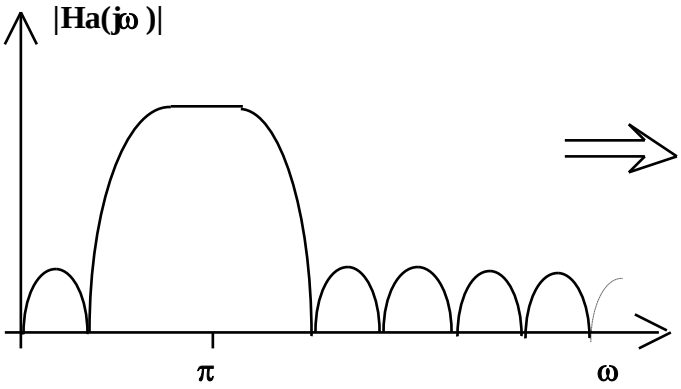


Fig.(a): Analogue Gain Response

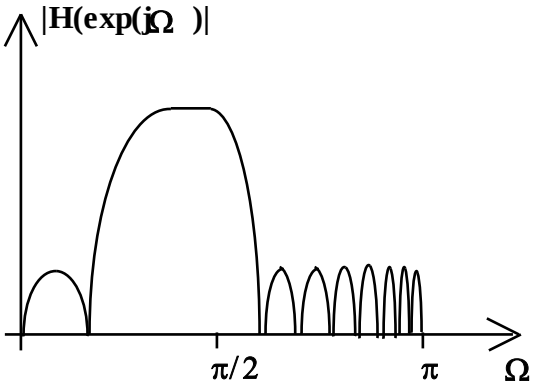


Fig. (b): Effect of Bilinear Transformation

'Prototype' analogue transfer function: Although the shape changes, we would like $G(\Omega)$ at its cut off Ω_C to the same as $G_a(\omega)$ at its cut-off frequency. If $G_a(\omega)$ is Butterworth, it is -3dB at its cut-off frequency. So we would like $G(\Omega)$ to be -3 dB at its cut-off Ω_C.

Achieved if the analogue prototype is designed to have its cut-off frequency at $\omega_C = 2 \tan(\Omega_C/2)$.

ω_C is then called the 'pre-warped' cut-off frequency.

Designing the analogue prototype with cut-off frequency 2 tan(Ω_C/2) guarantees that the digital filter will have its cut-off at Ω_C.

Design of a 2nd order IIR low-pass digital filter by the bilinear transform method ('by hand')

Let the required cut-off frequency $\Omega_C = \pi/4$ radians/sample. We need a prototype transfer function $H_a(s)$ for a 2nd order analogue Butterworth low-pass filter with 3 dB cut-off at ω_C = 2tan(Ω_C/2) = 2 tan(π/8) radians/second. Therefore, $\omega_C = 2\tan(\pi/8) = 0.828$. It is well known by analogue filter designers that the transfer function for a 2nd order Butterworth low-pass filter with cut-off frequency ω=1 radian/second is:

$$H_a(s) = \frac{1}{1+(\sqrt{2})s+s^2}$$

When the cut-off frequency is $\omega = \omega_C$ rather than $\omega = 1$, the second order expression for H(s) becomes:

$$H_a(s) = \frac{1}{1+\sqrt{2}(s/\omega_C)+(s/\omega_C)^2}$$

Replacing s by jω and taking the modulus of this expression gives $G(\omega) = 1/\sqrt{[1+(\omega/\omega_C)^{2n}]}$ with n=2. This is the 2nd order Butterworth low-pass gain-response approximation. Deriving the above expression for $H_a(s)$, and corresponding expressions for higher orders, is not part of our syllabus. It will not be necessary since MATLAB will take care of it.

Setting $\omega_C = 0.828$ in this formula, then replacing s by 2(z-1)/(z+1) gives us H(z) for the required IIR digital filter. You can check this 'by hand', but fortunately MATLAB does all this for us.

Example :

Using MATLAB, design a second order Butterworth-type IIR low-pass filter with $\Omega_c = \pi / 4$.

Solution:

[a b] = butter(2, 0.25)

a = [0.098 0.196 0.098]

b = [1 -0.94 0.33]

The required expression for H(z) is

$$H(z) = \frac{0.098 + 0.196\, z^{-1} + 0.098\, z^{-2}}{1 - 0.94\, z^{-1} + 0.33 z^{-2}}$$

$$H(z) = 0.098\left(\frac{1 + 2z^{-1} + z^{-2}}{1 - 0.94z^{-1} + 0.33z^{-2}}\right)$$

which may be realised by the signal flow graph in fig 5.18. Note the saving of two multipliers by using a multipler to scale the input by 0.098.

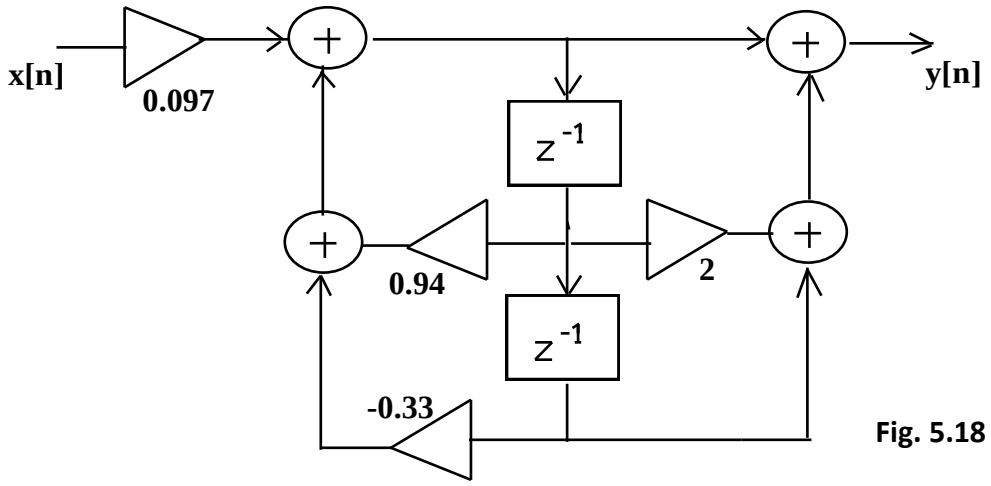


Fig. 5.18

Higher order IIR digital filters:

Recursive filters of order greater than two are highly sensitive to quantisation error and overflow. It is normal, therefore, to design higher order IIR filters as cascades of bi-quadratic sections. MATLAB does not do this directly as demonstrated by Example 5.8.

Example : Design a 4th order Butterworth-type IIR low-pass digital filter is needed with 3dB cut-off at one sixteenth of the sampling frequency f_S.

Solution: Relative cut-off frequency is $\pi/8$. The MATLAB command below produces the arrays a and b with the numerator and denominator coefficients for the 4th order system function H(z).

[a b] = butter(4, 0.125)

Output produced by MATLAB is:

a = 0.00093 0.0037 0.0056 0.0037 0.00093

b = 1 -2.9768 3.4223 -1.7861 0.3556

The system function is therefore as follows:

$$H(z)=\left(\frac{0.00093+0.0037z^{-1}+.0056z^{-2}+.0037z^{-3}+0.00093z^{-4}}{1-2.977z^{-1}+3.422z^{-2}-1.786z^{-3}+0.3556z^{-4}}\right)$$

This corresponds to the '4th order 'direct form' signal flow graph shown below.

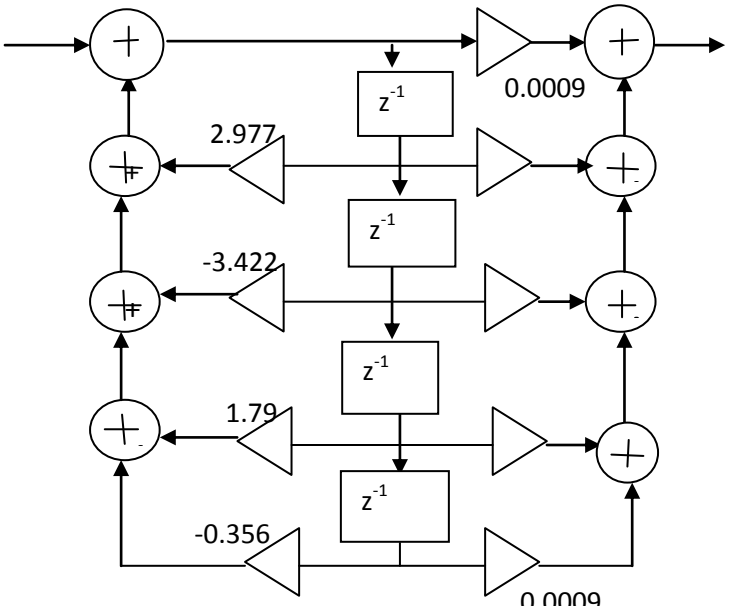


Figure : A 4th order 'direct form II' realisation (not commonly used)

Higher order IIR digital filters are generally not implemented like this. Instead, they are implemented as cascaded biquad or second order sections (SOS). Fortunately MATLAB can transform the 'direct form' coefficients to second order section (SOS) coefficients using a 'Signal Processing Toolbox' function 'tf2sos' as follows:

[a b] = butter(4, 0.125)

[sos G] = tf2sos(a,b)

Executing these statements gives the following response:

[a b] = butter(4, 0.125)

a = [0.0009 0.0037 0.0056 0.0037 0.0009]

b = [1 -2.9768 3.4223 -1.7861 0.3556]

[sos G] = tf2sos(a,b)

sos = [1 2 1 1 -1.365 0.478

1 2 1 1 -1.612 0.745]

G = 0.00093

This produces a 2-dimensional array 'sos' containing two sets of biquad coefficients and a 'gain' constant G. A mathematically correct system function based on this data is as follows:

$$H(z) = 0.00093\left(\frac{1+2z^{-1}+z^{-2}}{1-1.365z^{-1}+0.478z^{-2}}\right)\left(\frac{1+2z^{-1}+z^{-2}}{1-1.612z^{-1}0.745z^{-2}}\right)$$

In practice, especially in fixed point arithmetic, the effect of G is often distributed among the two sections. Noting that 0.033 x 0.028 $\approx$ 0.00093, and noting also that the two sections can be in either order, an alternative expression for H(z) is as follows:

$$H(z) = 0.033\left(\frac{1+2z^{-1}+z^{-2}}{1-1.612z^{-1}0.745z^{-2}}\right)0.028\left(\frac{1+2z^{-1}+z^{-2}}{1-1.365z^{-1}+0.478z^{-2}}\right)$$

This alternative expression for H(z) may be realised in the form of cascaded bi-quadratic sections as shown in fig 5.20.

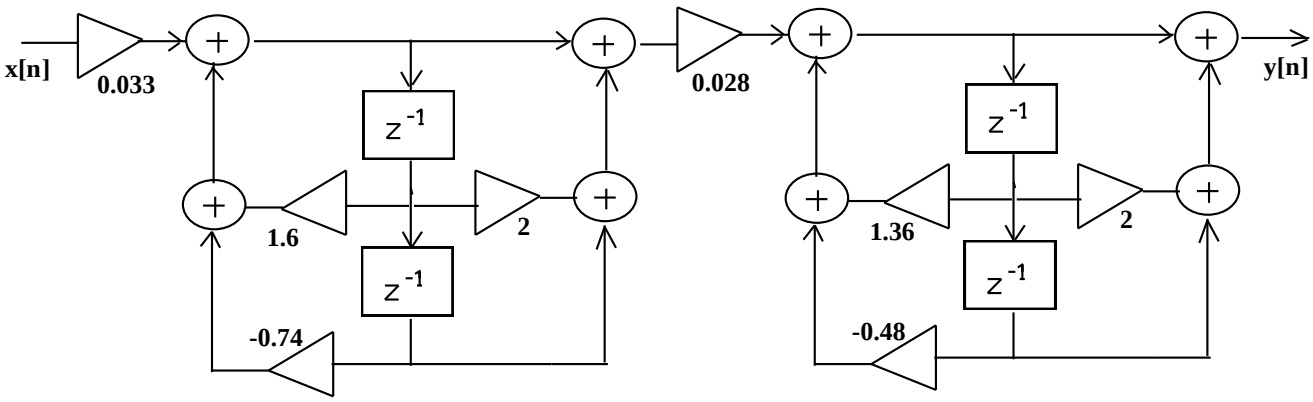


Fig. : Fourth order IIR Butterworth LP filter with cut-off *fs*/16

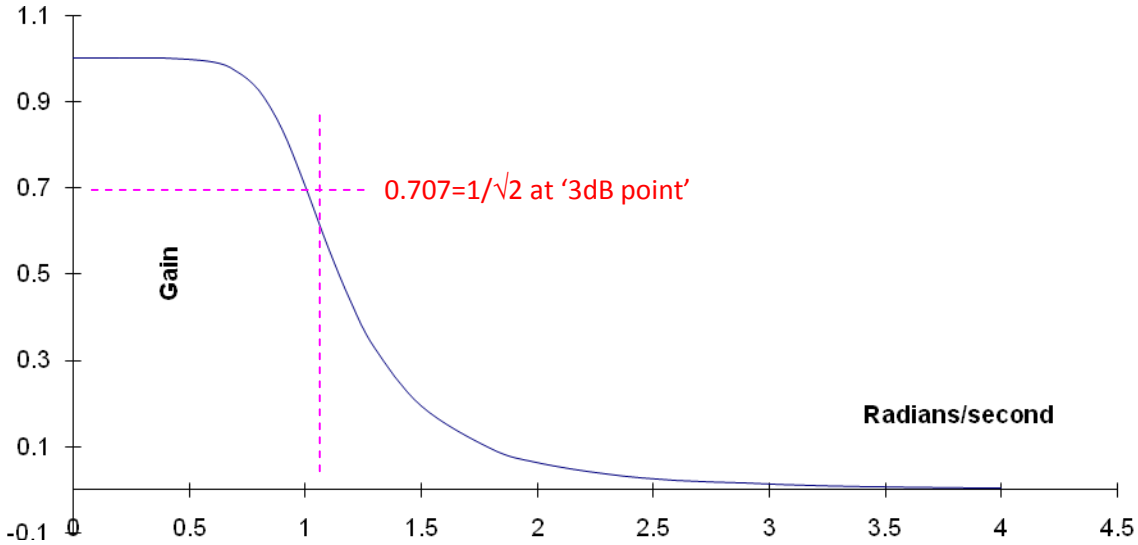


Fig. (a) Analogue 4th order Butterworth LP gain response

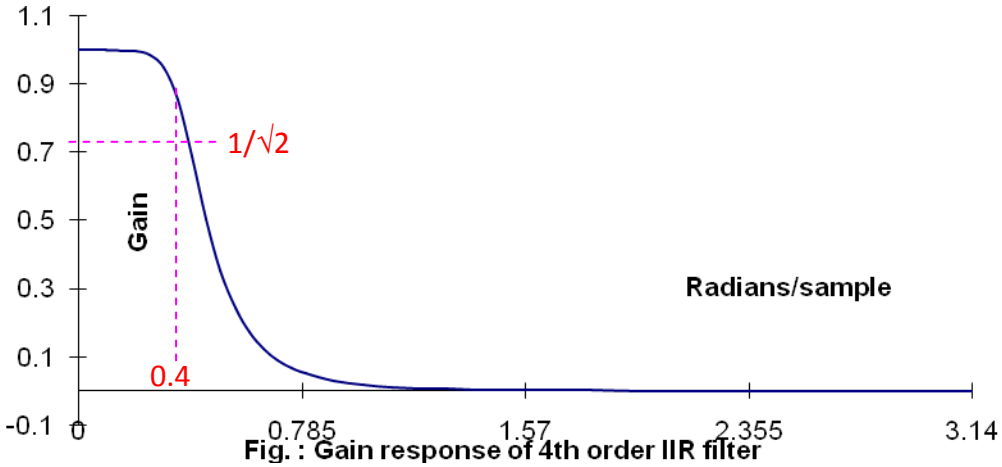


Fig. : Gain response of 4th order IIR filter

Fig. (a) shows the 4th order Butterworth low-pass gain response:

$$G(\omega) = \frac{1}{\sqrt{(1+\omega^8)}}$$

(with cut-off frequency normalised to 1) as used by MATLAB as a prototype. Fig 5.21(b) shows the gain-response of the derived digital filter which, like the analogue filter, is 1 at zero frequency and 0.707 (-3dB) at the cut-off frequency ($\pi/8 \approx 0.39$ radians/sample). Note however that the analogue gain approaches 0 as $\omega \to \infty$ whereas the gain of the digital filter becomes exactly zero at $\Omega = \pi$. The shape of the Butterworth gain response is 'warped' by the bilinear transformation. However, the 3dB point occurs exactly at Ω_c for the digital filter, and the cut-off rate becomes sharper and sharper as $\Omega \to \pi$ because of the compression as $\omega \to \infty$.

IIR digital high-pass band-pass and band-stop filter design:

The bilinear transformation may be applied to analogue system functions which are high-pass, band-pass or band-stop to obtain digital filter equivalents. For example a 'high-pass' digital filter may be designed as illustrated below:

Example : Design a 4th order high-pass IIR filter with cut-off frequency *fs*/16.

Solution: Execute the following MATLAB commands and proceed as for low-pass

```
[a b] = butter(4,0.125,'high');

freqz(a,b);

[sos G] = tf2sos(a,b)
```

Wide-band band-pass and band-stop filters ($f_U >> 2f_L$) may be designed by cascading low-pass and high-pass sections, but 'narrow band' band-pass/stop filters (f_U not $>> 2f_L$) will not be very accurate if this cascading approach is used. The MATLAB band-pass approach always works, i.e. for narrowband and wideband. A possible source of confusion is that specifying an order '2' produces what many people (including me, Barry) would call a 4th order IIR digital filter. The design process carried out by 'butter' involves the design of a low-pass prototype and then applying a low-pass to band-pass transformation which doubles the complexity. The order specified is the order of the prototype. So if we specify 2nd order for band-pass we get a 4th order system function which can be re-expressed (using tf2sos) as TWO biquad sections.

Example : Design a 2nd (4th)order bandpass filter with $\Omega_L = \pi/4$, $\Omega_U = \pi/2$.

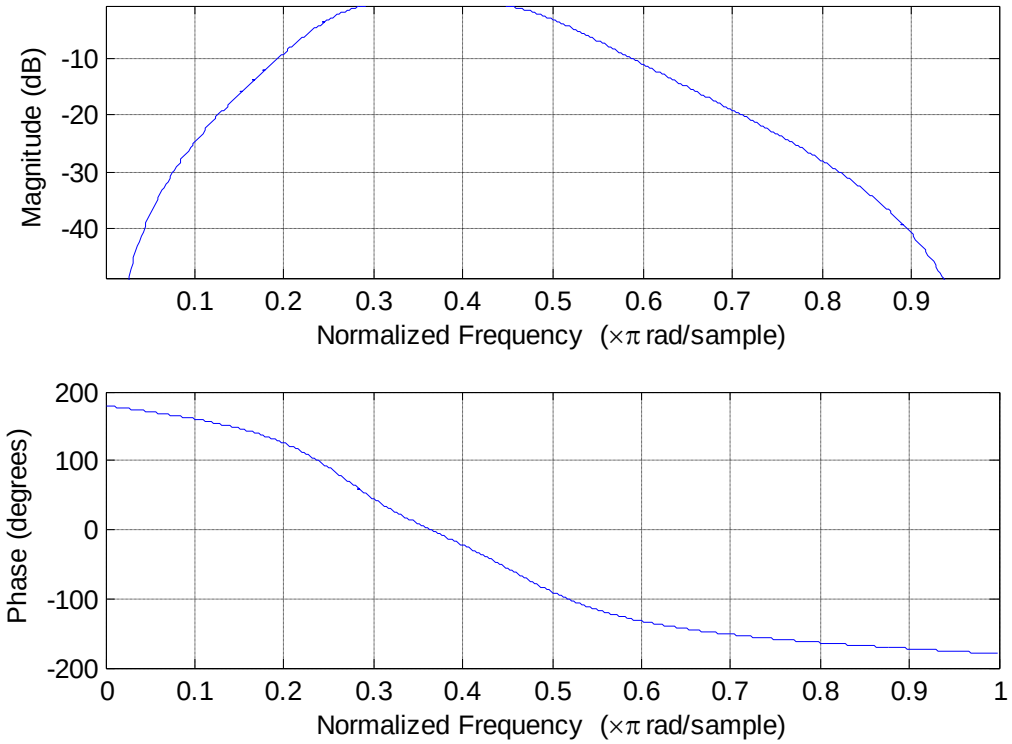
Solution: Execute the following MATLAB statements:

```
[a b] = butter(2,[0.25 0.5])

freqz(a,b);
```

```
[sos G] = tf2sos(a,b)
```

MATLAB output:is:

```
a =  0.098  0   -0.195  0  0.098

b = 1   -1.219    1.333   -0.667    0.33

sos =  1    2    1    1   -0.1665    0.5348

       1   -2    1    1   -1.0524    0.6232

G = 0.098
```



Example : Design a 4th (8th)order bandpass filter with $\omega_L = \pi/4$, $\omega_u = \pi/2$.

Solution: Execute the following MATLAB statements

```
[a b] = butter(4,[0.25 0.5])

freqz(a,b);  axis([0 1 -40 0]);

[sos G] = tf2sos(a,b)
```

to obtain the MATLAB output:

a = 0.01 0 -0.041 0 0.061 0 -0.041 0 0.01

b = 1 -2.472 4.309 -4.886 4.477 -2.914 1.519 -0.5 0.12

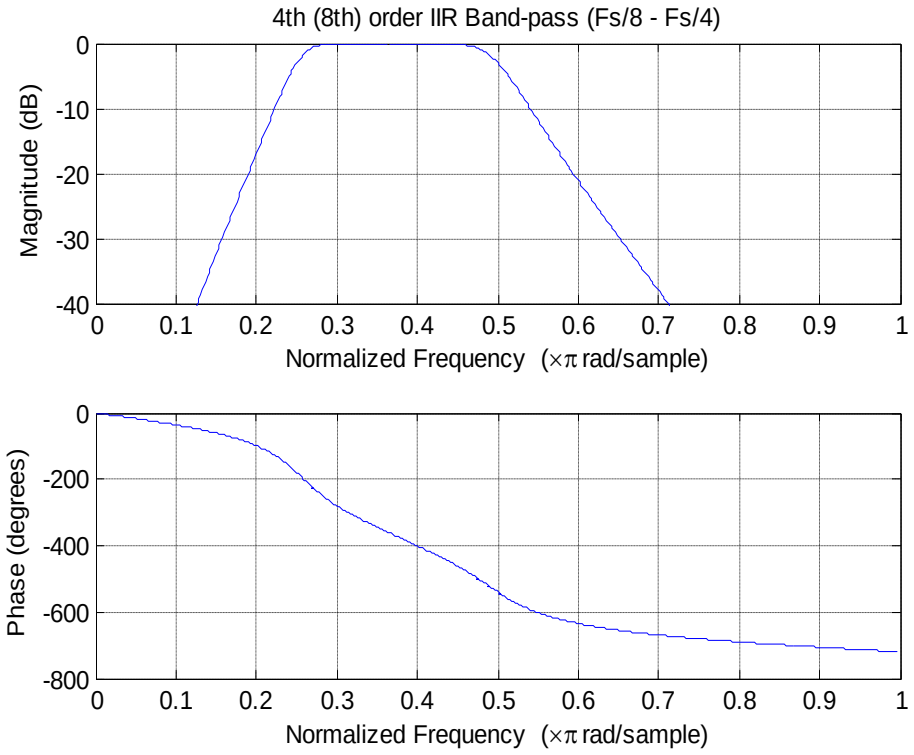
sos =1 2 1 1 -0.351 0.428

1 -2. 1 1 -0.832 0.49

1 2. 1 1 -0.046 0.724

1 -2 1 1 -1.244 0.793

G = 0.01



Example : Design a 4th (8th)order band-stop filter with $\omega_L = \pi/4$, $\omega_u = \pi/2$.

Solution: Execute the following MATLAB statements

[a b] = butter(4,[0.25 0.5], 'stop')

freqz(a,b); axis([0 1 -40 0]);

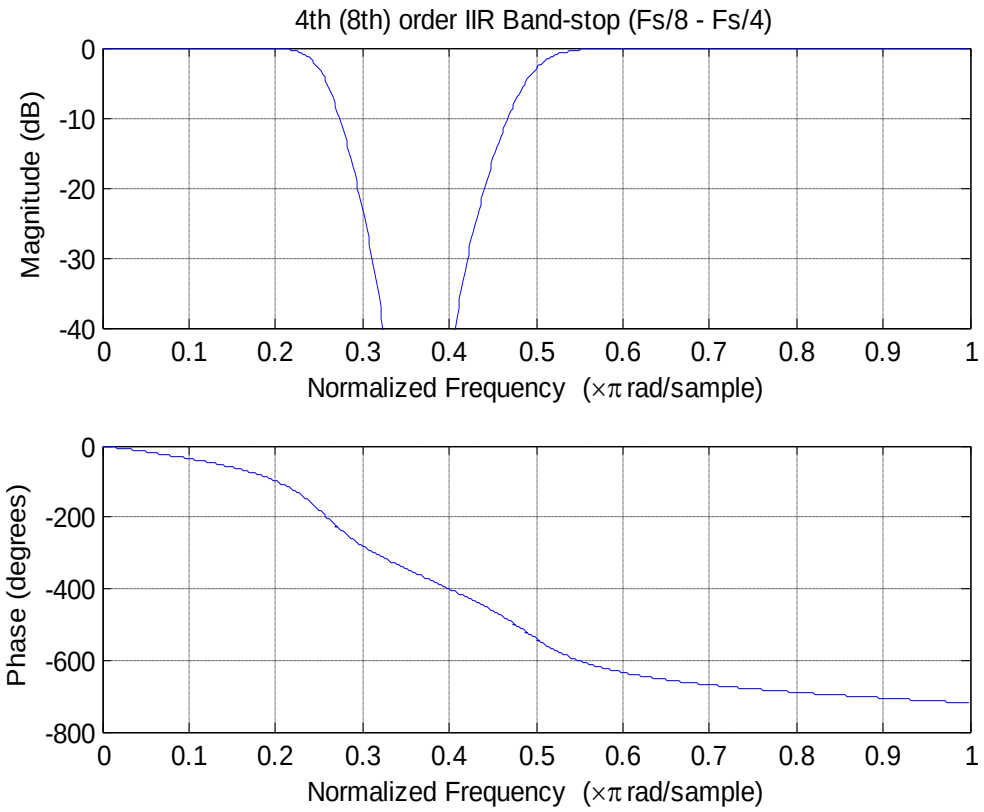
```
[sos G] = tf2sos(a,b)
```

to obtain the MATLAB output:

a = 0.347 -1.149 2.815 -4.237 5.1 -4.237 2.815 -1.149 0.347

b = 1 -2.472 4.309 -4.886 4.477 -2.914 1.519 -0.5 0.12

```
sos =   1  -0.828   1   1  -0.351   0.428
        1  -0.828   1   1  -0.832   0.49
        1  -0.828   1   1  -0.046   0.724
        1  -0.828   1   1  -1.244   0.793

G = 0.347
```



Comparison of IIR and FIR digital filters:

IIR type digital filters have the advantage of being economical in their use of delays, multipliers and adders. They have the disadvantage of being sensitive to coefficient round-off inaccuracies and the effects of overflow in fixed point arithmetic. These effects can lead to instability or serious distortion. Also, an IIR filter cannot be exactly linear phase.

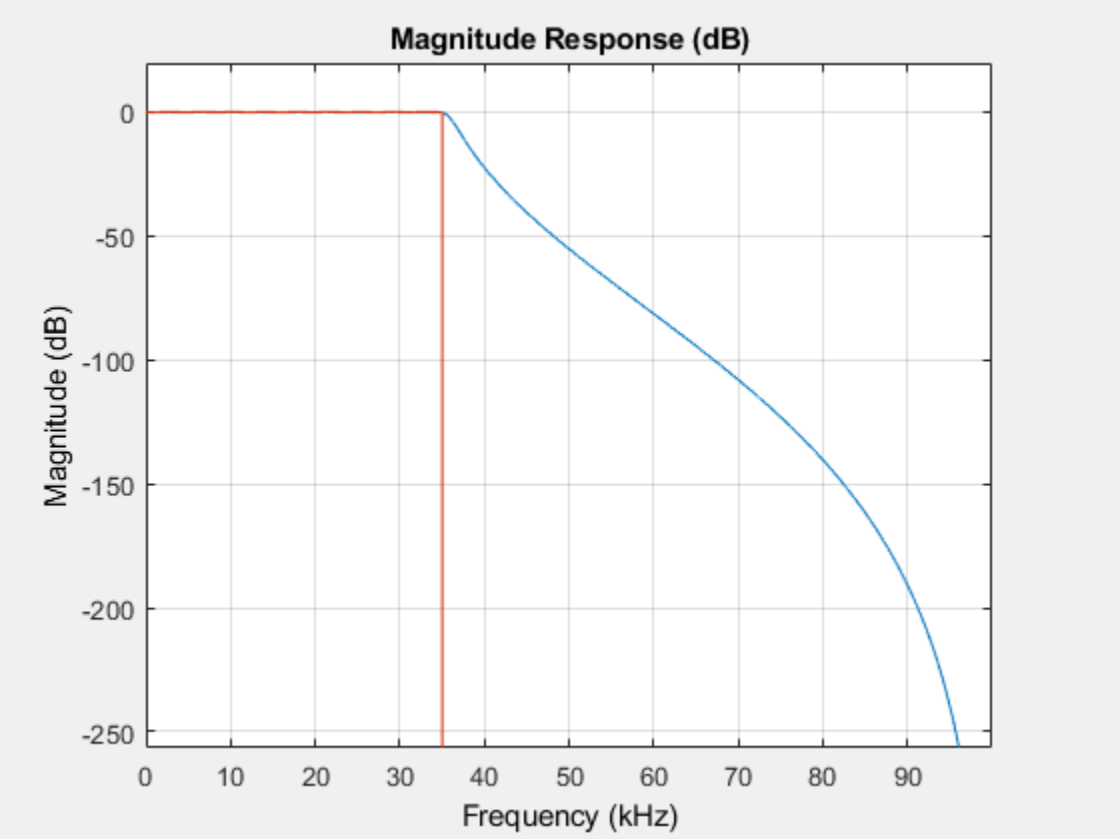
FIR filters may be realised by non-recursive structures which are simpler and more convenient for programming especially on devices specifically designed for digital signal processing. These structures are always stable, and because there is no recursion, round-off and overflow errors are easily controlled. A FIR filter can be exactly linear phase. The main disadvantage of FIR filters is that large orders can be required to perform fairly simple filtering tasks.

Lowpass IIR Filter

Try This Example

Design a lowpass IIR filter with order 8, passband frequency 35 kHz, and passband ripple 0.2 dB. Specify a sample rate of 200 kHz. Visualize the magnitude response of the filter. Use it to filter a 1000-sample random signal.

```
lpFilt = designfilt('lowpassiir','FilterOrder',8, ...
         'PassbandFrequency',35e3,'PassbandRipple',0.2, ...
         'SampleRate',200e3);
fvtool(lpFilt)
```




```
dataIn = randn(1000,1);
dataOut = filter(lpFilt,dataIn);
```

Output the filter coefficients, expressed as second-order sections.

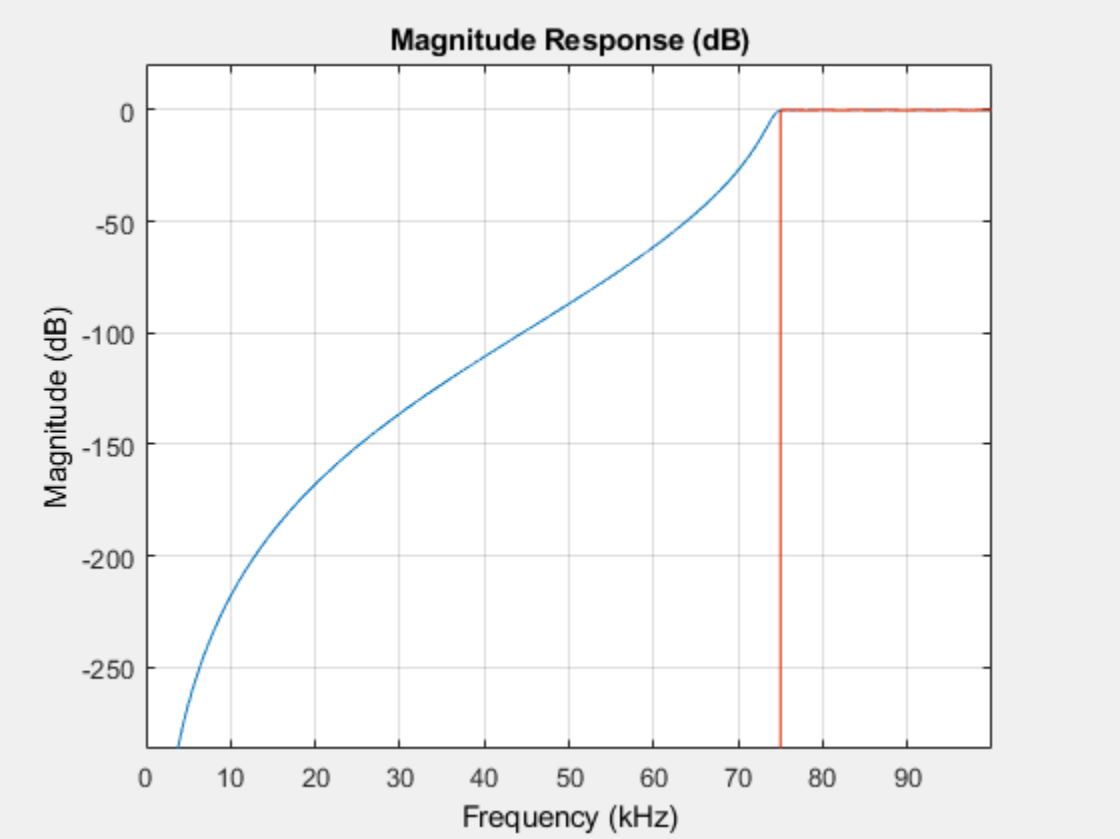
```
sos = lpFilt.Coefficients
sos = 4×6

    0.2666    0.5333    0.2666    1.0000   -0.8346    0.9073
    0.1943    0.3886    0.1943    1.0000   -0.9586    0.7403
    0.1012    0.2023    0.1012    1.0000   -1.1912    0.5983
    0.0318    0.0636    0.0318    1.0000   -1.3810    0.5090
```

Highpass IIR Filter

Try This Example

Design a highpass IIR filter with order 8, passband frequency 75 kHz, and passband ripple 0.2 dB. Specify a sample rate of 200 kHz. Visualize the filter's magnitude response. Apply the filter to a 1000-sample vector of random data.

```
hpFilt = designfilt('highpassiir','FilterOrder',8, ...
         'PassbandFrequency',75e3,'PassbandRipple',0.2, ...
         'SampleRate',200e3);
fvtool(hpFilt)
```

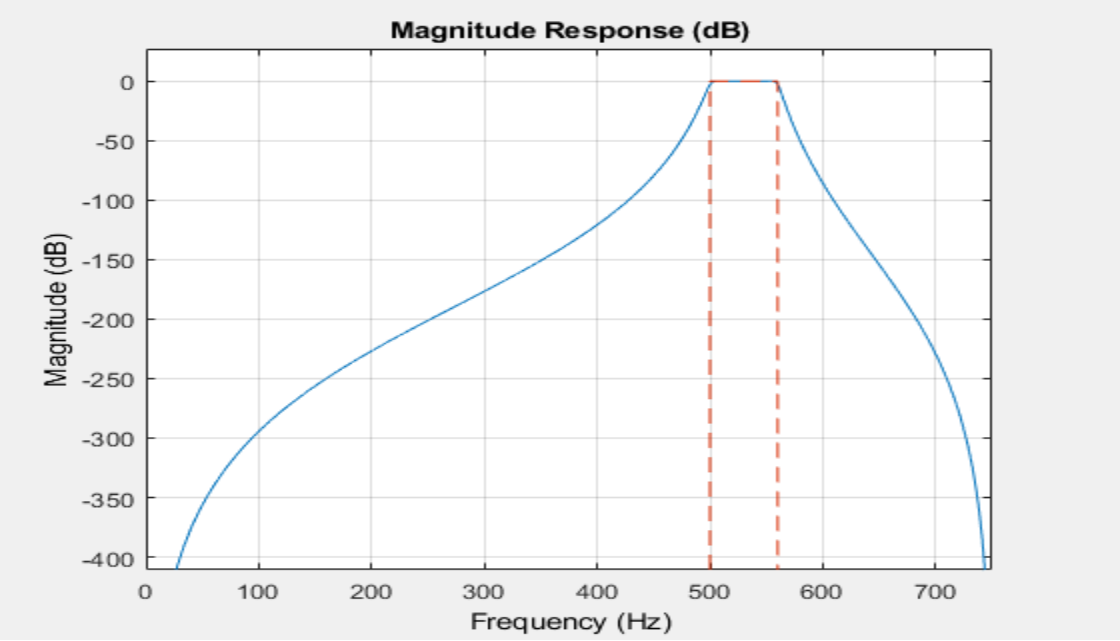



```
dataIn = randn(1000,1);
dataOut = filter(hpFilt,dataIn);
```

Bandpass IIR Filter

Try This Example:Design a 20th-order bandpass IIR filter with lower 3-dB frequency 500 Hz and higher 3-dB frequency 560 Hz. The sample rate is 1500 Hz. Visualize the frequency response of the filter. Use it to filter a 1000-sample random signal.

```
bpFilt = designfilt('bandpassiir','FilterOrder',20, ...
       'HalfPowerFrequency1',500,'HalfPowerFrequency2',560, ...
       'SampleRate',1500);
fvtool(bpFilt)
```

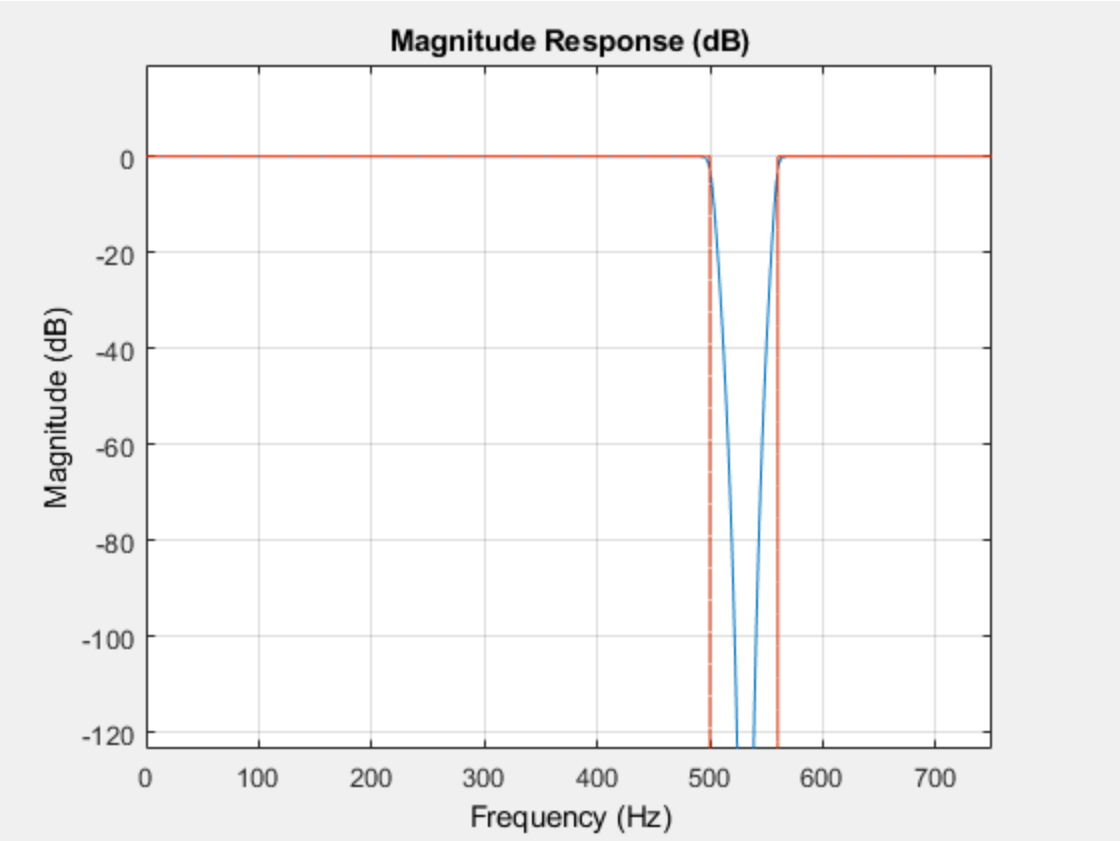



```
dataIn = randn(1000,1);
dataOut = filter(bpFilt,dataIn);
```

Bandstop IIR Filter

Design a 20th-order bandstop IIR filter with lower 3-dB frequency 500 Hz and higher 3-dB frequency 560 Hz. The sample rate is 1500 Hz. Visualize the magnitude response of the filter. Use it to filter 1000 samples of random data.

```
bsFilt = designfilt('bandstopiir','FilterOrder',20, ...
    'HalfPowerFrequency1',500,'HalfPowerFrequency2',560, ...
    'SampleRate',1500);
fvtool(bsFilt)
```


Magnitude Response (dB)
Magnitude (dB)
0
-20
-40
-60
-80
-100
-120
0
100
200
300
400
500
600
700
Frequency (Hz)

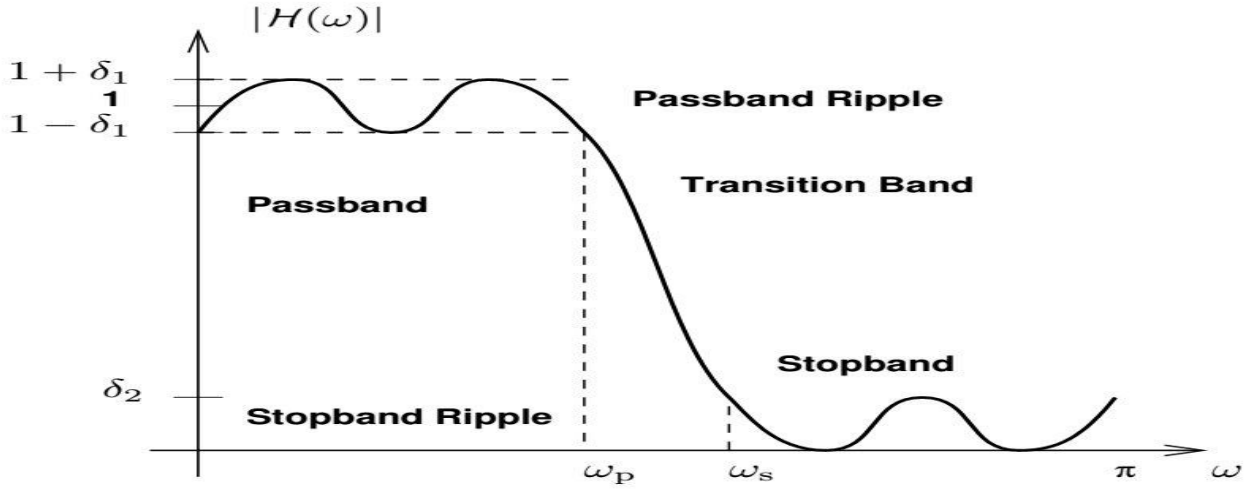
UNIT-IV

DESIGN OF FIR FILTERS

Introduction

Filters are used in a wide variety of applications. Most of the time, the final aim of using a filter is to achieve a better frequency selectivity on the spectrum of the input signal. At this point of time, it is required to reviewing the frequency response of a practical filter. The below Figure (A) shows an example of a practical low pass filter.

In this example, frequency components in the pass band, from DC to ω_P will pass through the filter almost with no attenuation. The components in the stop band, above ω_S will experience significant attenuation. Note that the frequency response of a practical filter cannot be absolutely flat in the pass band or in the stop band. As shown in Figure (A), some ripples will be unavoidable and the transition band, ωp<ω<ωs cannot be infinitely sharp in practice.



Digital filter design involves four steps:

1) Determining specifications

First, we need to determine what specifications are required. This step completely depends on the application. This information is necessary to find the filter with minimum order for this application.

2) **Finding a transfer function**

With design specifications known, we need to find a transfer function which will provide the required filtering. The rational transfer function of a digital filter is as given below.

$$H(z) = \frac{\sum_{k=0}^{M-1} b_k z^{-k}}{\sum_{k=0}^{N-1} a_k z^{-k}}$$

3) Choosing a realization structure

Now that H(z) is known, we should choose the realization structure. In other words, there are many systems which can give the obtained transfer function and we must choose the appropriate one. For example, any of the direct form I, II, cascade, parallel, transposed, or lattice forms can be used to realize a particular transfer function. The main difference between the aforementioned realization structures is their sensitivity to using a finite length of bits. Note that in the final digital system, we will use a finite length of bits to represent a signal or a filter coefficient. Some realizations, such as direct forms, are very sensitive to quantization of the coefficients. However, cascade and parallel structures show smaller sensitivity and are preferred.

4) **Implementing the filter**

After deciding on what realization structure to use, we should implement the filter. You have a couple of options for this step: a software implementation (such as a MATLAB or C code) or a hardware implementation (such as a DSP, a microcontroller, or an ASIC).

it is necessary to take into account all fundamental characteristics of a signal to be filtered as these are very important when deciding which filter to use. In most cases, it is only one characteristic that really matters and it is whether it is necessary that filter has linear phase characteristic or not. it is necessary that a filter has linear phase characteristic to prevent loosing important information.When a signal to be filtered is analysed in this way, it is easy to decide which type of digital filter is best to use. Accordingly, if the phase characteristic is of the essence, FIR filters should be used as they have linear phase characteristic. Such filters are of higher order and more complex, therefore. The FIR Filters can be easily designed to have perfectly linear Phase. These filters can be realized recursively and Non-recursively. There are greater flexibility to control the Shape of their Magnitude response. Errors due to round off noise are less severe in FIR Filters, mainly because Feed back is not used

An FIR digital filter of order M may be implemented by programming the signal-flow-graph shown below. Its difference equation is:

y[n] = a0x[n] + a1x[n-1] + a2x[n-2] + ... + aMx[n-M]

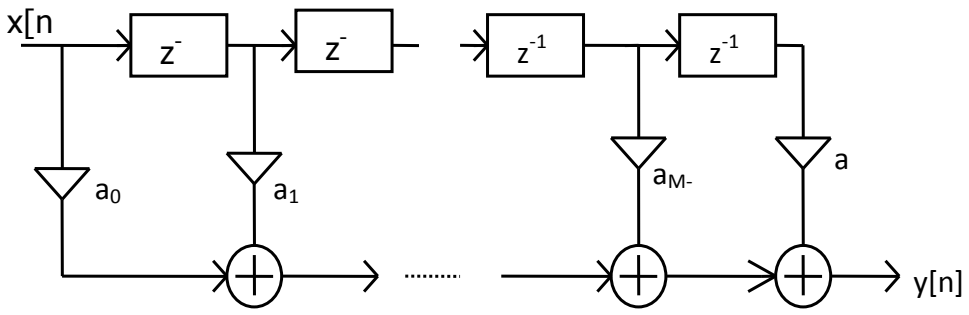


Fig. 4.1

Its impulse-response is {..., 0, ..., a0, a1, a2,..., aM, 0, ...} and its frequency-response is the DTFT of the impulse-response, i.e.

$$H(e^{j\Omega}) = \sum_{n=-\infty}^{\infty} h[n]e^{-j\Omega n} = \sum_{n=0}^{M} a_n e^{-j\Omega n}$$

Now consider the problem of choosing the multiplier coefficients. a0, a1,..., aM such that H(ejΩ) is close to some desired or target frequency-response H'(ejΩ) say. The inverse DTFT of H'(ejΩ) gives the required impulse-response :

$$h'[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} H'(e^{j\Omega})e^{j\Omega n}d\Omega$$

The methodology is to use the inverse DTFT to get an impulse-response {h'[n]} & then realise some approximation to it Note that the DTFT formula is an integral, it has complex numbers and the range of integration is from -π to π, so it involves negative frequencies.

What about the negative frequencies?

Examine the DTFT formula for H(ejΩ).

$$H(e^{j\Omega}) = \sum_{n=-\infty}^{\infty} h[n]e^{-j\Omega n} \qquad \therefore \quad H(e^{-j\Omega}) = \sum_{n=-\infty}^{\infty} h[n]e^{j\Omega n}$$

If h[n] real then h[n]ejΩ is complex-conjugate of h[n]e-jΩ. Adding up terms gives H(e-jΩ) as complex conj of H(ejΩ).

G(Ω) = G(-Ω) since G(Ω) = |H(ejΩ)| & G(-Ω) = H(e-jΩ)|

Because of the range of integration (-π to π) of the DTFT formula, it is common to plot graphs of G(Ω) and ϕ(Ω) over the frequency range -π to π rather than 0 to π. As G(Ω) = G(-Ω) for a real filter the gain-response will always be symmetric about Ω=0.

Features of FIR Filter

1. FIR filter always provides linear phase response. This specifies that the signals in the pass band will suffer no dispersion Hence when the user wants no phase distortion, then FIR filters

are preferable over IIR. Phase distortion always degrade the system performance. In various applications like speech processing, data transmission over long distance FIR filters are more preferable due to this characteristic.

2. FIR filters are most stable as compared with IIR filters due to its non feedback nature.

3. Quantization Noise can be made negligible in FIR filters. Due to this sharp cutoff FIR filters can be easily designed.

4. Disadvantage of FIR filters is that they need higher ordered for similar magnitude response of IIR filters.

Difference equation of FIR filter of length M is given as

$$y(n)=\sum_{k=0}^{M-1} b_k\, x(n-k) \qquad (1)$$

And the coefficient b_k are related to unit sample response as $H(n) = bn$ for $0 \le n \le M-1$,

$= 0$ otherwise

We can expand this equation as $Y(n)= b0\ x(n) + b1\ x(n-1) + \ldots\ldots\ldots + bM-1\ \ x(n-M+1)$ (2)

System is stable only if system produces bounded output for every bounded input. This is stability definition for any system. Here h(n)={b0, b1, b2, } of the FIR filter are stable. Thus y(n) is bounded if input x(n) is bounded. This means FIR system produces bounded output for every bounded input. Hence FIR systems are always stable.

The main features of FIR filter are,

- They are inherently stable
- Filters with linear phase characteristics can be designed
- Simple implementation – both recursive and non-recursive structures possible
- Free of limit cycle oscillations when implemented on a finite-word length digital system

Disadvantages:

- Sharp cutoff at the cost of higher order
- Higher order leading to more delay, more memory and higher cost of implementation

Of these, the linear phase property is probably the most important. A filter is said to have a generalised linear phase response if its frequency response can be expressed in the form

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

where α and β are constants, and $A(e^{j\omega})$ is a real function of ω. If this is the case, then

- If A is positive, then the phase is

$$\measuredangle H(e^{j\omega}) = \beta - \alpha\omega.$$

  If A is negative, then

$$\measuredangle H(e^{j\omega}) = \pi + \beta - \alpha\omega.$$

In either case, the phase is a linear function of ω.

It is common to restrict the filter to having a real-valued impulse response $h[n]$, since this greatly simplifies the computational complexity in the implementation of the filter.

A FIR system has linear phase if the impulse response satisfies either the even symmetric condition

$$h[n] = h[N-1-n],$$

or the odd symmetric condition

$$h[n] = -h[N-1-n].$$

The system has different characteristics depending on whether N is even or odd. Furthermore, it can be shown that all linear phase filters must satisfy one of these conditions. Thus there are exactly four types of linear phase filters.

Consider for example the case of an odd number of samples in $h[n]$, and even symmetry. The frequency response for $N = 7$ is

$$\begin{aligned} H(e^{j\omega}) &= \sum_{n=0}^{6} h[n]e^{-j\omega n} \\ &= h[0] + h[1]e^{-j\omega} + h[2]e^{-j2\omega} + h[3]e^{-j3\omega} + h[4]e^{-j4\omega} \\ &\qquad + h[5]e^{-j5\omega} + h[6]e^{-j6\omega} \\ &= e^{-j3\omega}(h[0]e^{j3\omega} + h[1]e^{j2\omega} + h[2]e^{j\omega} + h[3] + h[4]e^{-j\omega} \\ &\qquad + h[5]e^{-j2\omega} + h[6]e^{-j3\omega}). \end{aligned}$$

The specified symmetry property means that $h[0] = h[6]$, $h[1] = h[5]$, and $h[2] = h[4]$, so

$$\begin{aligned} H(e^{j\omega}) &= e^{-j3\omega}(h[0](e^{j3\omega} + e^{-j3\omega}) + h[1](e^{j2\omega} + e^{-j2\omega}) \\ &\quad + h[2](e^{j\omega} + e^{-j\omega}) + h[3]) \\ &= e^{-j3\omega}(2h[0]\cos(3\omega) + 2h[1]\cos(2\omega) + 2h[2]\cos(\omega)) \\ &= e^{-j3\omega}\sum_{n=0}^{3} a[n]\cos(\omega n), \end{aligned}$$

where $a[0] = h[3]$, and $a[n] = 2h[3-n]$ for $n = 1, 2, 3$. The resulting filter clearly has a linear phase response for real $h[n]$. It is quite simple to show that in general for odd values of N the frequency response is

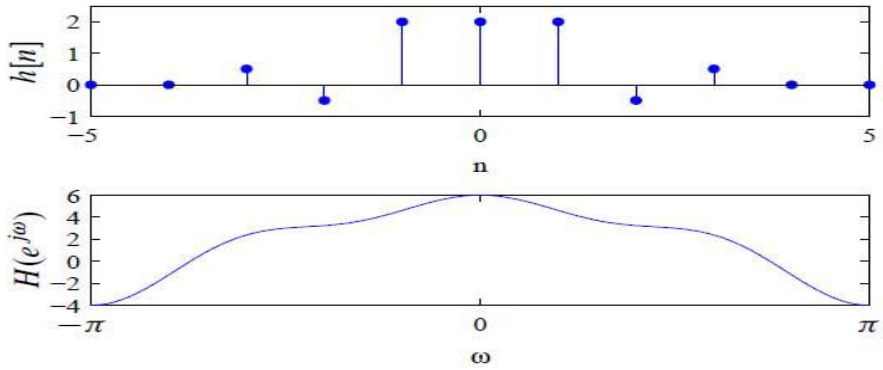
$$H(e^{j\omega}) = e^{-j\omega(N-1)/2}\sum_{n=0}^{(N-1)/2} a[n]\cos(\omega n),$$

for a set of real-valued coefficients $a[0], \ldots, a[(N-1)/2]$. As different values for $a[n]$ are selected, different linear-phase filters are obtained.

The cases of N odd and $h[n]$ antisymmetric are similar to that presented, and the frequency responses are summarised in the following table:

Symmetry	N	$H(e^{j\omega})$	Type
Even	Odd	$e^{-j\omega(N-1)/2}\sum_{n=0}^{(N-1)/2} a[n]\cos(\omega n)$	1
Even	Even	$e^{-j\omega(N-1)/2}\sum_{n=1}^{N/2} b[n]\cos(\omega(n-1/2))$	2
Odd	Odd	$e^{-j[\omega(N-1)/2-\pi/2]}\sum_{n=0}^{(N-1)/2} a[n]\sin(\omega n)$	3
Odd	Even	$e^{-j[\omega(N-1)/2-\pi/2]}\sum_{n=1}^{N/2} b[n]\sin(\omega(n-1/2))$	4

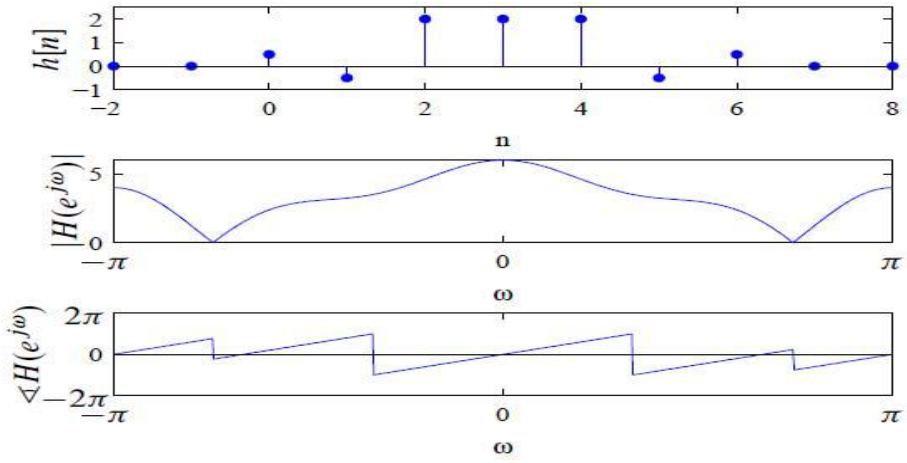
Recall that even symmetry implies $h[n] = h[N-1-n]$ and odd symmetry $h[n] = -h[N-1-n]$. Examples of filters satisfying each of these symmetry conditions are:



Recall from the properties of the Fourier transform this filter has a real-valued frequency response $A(e^{j\omega})$. Delaying this impulse response by $(N-1)/2$ results in a causal filter with frequency response

$$H(e^{j\omega}) = A(e^{j\omega})e^{-j\omega(N-1)/2}.$$

This filter therefore has linear phase.



Symmetric and Anti-symmetric FIR filters

Unit sample response of FIR filters is symmetric if it satisfies following condition.

h(n)= h(M-1-n), for n=0,1,2…….M-1 2.

Unit sample response of FIR filters is Anti-symmetric if it satisfies following condition

h(n)= -h(M-1-n) for n=0,1,2.

FIR filters giving out Linear Phase characteristics: Symmetry in filter impulse response will ensure linear phase An FIR filter of length M with i/p x(n) & o/p y(n) is described by the difference_equation

$$y(n)= b_0\, x(n) + b_1\, x(n-1)+\ldots\ldots+b_{M-1}\, x(n-(M-1)) = \sum_{k=0}^{M-1} b_k\, x(n-k) \qquad -(1)$$

Alternatively. it can be expressed in convolution form

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

i.e $b_k = h(k)$, k=0,1,….M-1

Choice of Symmetric and anti-symmetric unit sample response

When we have a choice between different symmetric properties, the particular one is picked up based on application for which the filter is used. The following points give an insight to this issue. • If h(n)=-h(M-1-n) and M is odd, Hr(w) implies that Hr(0)=0 & Hr(π)=0, consequently not suited for low pass and high pass filter. This condition is suited in Band Pass filter design.

• Similarly if M is even Hr(0)=0 hence not used for low pass filter

• Symmetry condition h(n)=h(M-1-n) yields a linear-phase FIR filter with non zero response at w = 0 if desired. Looking at these points, anti-symmetric properties are not generally preferred

Poles & Zeros of linear phase sequences:

The poles of any finite-length sequence must lie at z=0. The zeros of linear phase sequence must occur in conjugate reciprocal pairs. Real zeros at z=1 or z=-1 need not be paired (they form their own reciprocals), but all other real zeros must be paired with their reciprocals. Complex zeros on the unit circle must be paired with their conjugate (that form their reciprocals) and complex zeros anywhere else must occur in conjugate reciprocal quadruples. To identify the type of sequence from its pole-zero plot, all we need to do is check for the presence of zeros at z= □ and count their number. A type2 seq must have an odd number of zeros at z=-1, a type-3 seq must have an odd number of zeros at z=-1 and z=1, and type-4 seq must have an odd number of zeros at z=1. The no. of other zeros if present (at z=1 for type=1 and type-2 or z=-1 for type-1 or type-4) must be even.

Zeros of Linear Phase FIR Filters:

Consider the filter system function

$$H(z) = \sum_{n=o}^{M-1} h(n) z^{-n}$$

Expanding this equation

$$H(z) = h(0) + h(1)z^{-1} + h(2)z^{-2} \quad ++ h(M-2)z^{-(M-2)} + h(M-1)z^{-(M-1)}$$

$\sin ce\ for\ Linear-phase\ we\ need$

$h(n) = h(M-1-n)$ *i.e.,*

$h(0) = h(M-1);\ h(1) = h(M-2);\ldots\ldots h(M-1) = h(0);$

then

$$H(z) = h(M-1) + h(M-2)z^{-1} + \ldots\ldots + h(1)z^{-(M-2)} + h(0)z^{-(M-1)}$$

$$H(z) = z^{-(M-1)}[h(M-1)z^{(M-1)} + h(M-2)z^{(M-2)} + \ldots\ldots + h(1)z + h(0)]$$

$$H(z) = z^{-(M-1)}[\sum_{n=0}^{M-1} h(n)(z^{-1})^{-n}] = z^{-(M-1)} H(z^{-1})$$

This shows that if z = z1 is a zero then $z=z1^{-1}$ is also a zero
The different possibilities: 1. If z1 = 1 then $z1 = z1^{-1} = 1$ is also a zero implying it is one zero
2. If the zero is real and |z|<1 then we have pair of zeros
3. If zero is complex and |z|=1then and we again have pair of complex zeros.
4. If zero is complex and |z|≠1 then and we have two pairs of complex zeros

FIR Filter Design Methods

The various method used for FIR Filer design are as follows

1. Fourier Series method
2. Windowing Method
3. DFT method
4. Frequency sampling Method. (IFT Method)

Design of an FIR low-pass digital filter

Assume we require a low-pass filter whose gain-response approximates the ideal 'brick-wall' gain-response in Figure 4.2.

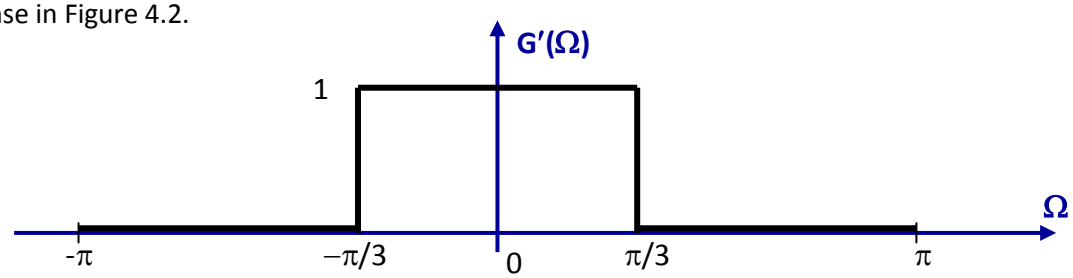


Fig. 4.2

If we take the phase-response $\phi'(\Omega)$ to be zero for all Ω, the required frequency-response is:-

$$H'(e^{j\Omega}) = G'(\Omega)e^{j\phi(\Omega)} = \begin{cases} 1 & : \ |\Omega| \le \pi/3 \\ 0 & : \ \pi/3 < |\Omega| < \pi \end{cases}$$

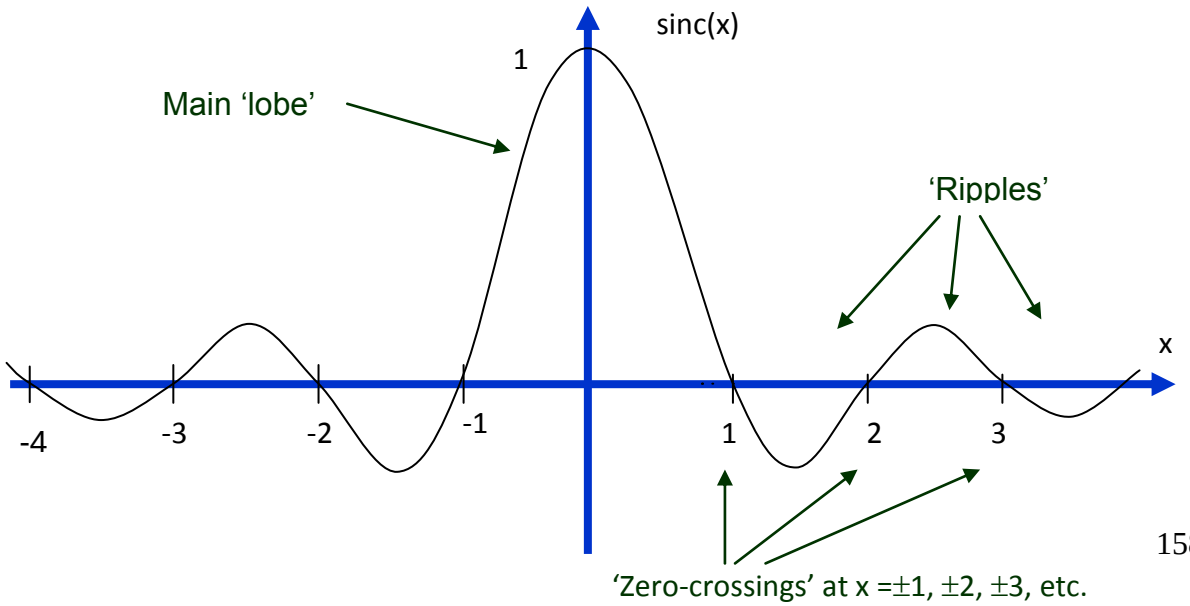
And by the inverse DTFT,

$$h'[n] = \frac{1}{2\pi}\int_{-\pi/3}^{\pi/3} 1 e^{j\Omega n} d\Omega = \begin{cases} 1/3 & : \ n = 0 \\ (1/n\pi)\sin(n\pi/3) & : \ n \neq 0 \end{cases}$$

$$= \ (1/3)\text{sinc}(n/3) \ \text{ for all n.}$$

$$\text{where } \text{sinc}(x) = \begin{cases} \dfrac{\sin(\pi x)}{\pi x} : x \neq 0 \\ 1 : x = 0 \end{cases}$$

A graph of sinc(x) against x is shown below:



Fig

The ideal impulse-response {h′[n]} with each sample h′[n] = (1/3) sinc(n/3) is therefore as follows:

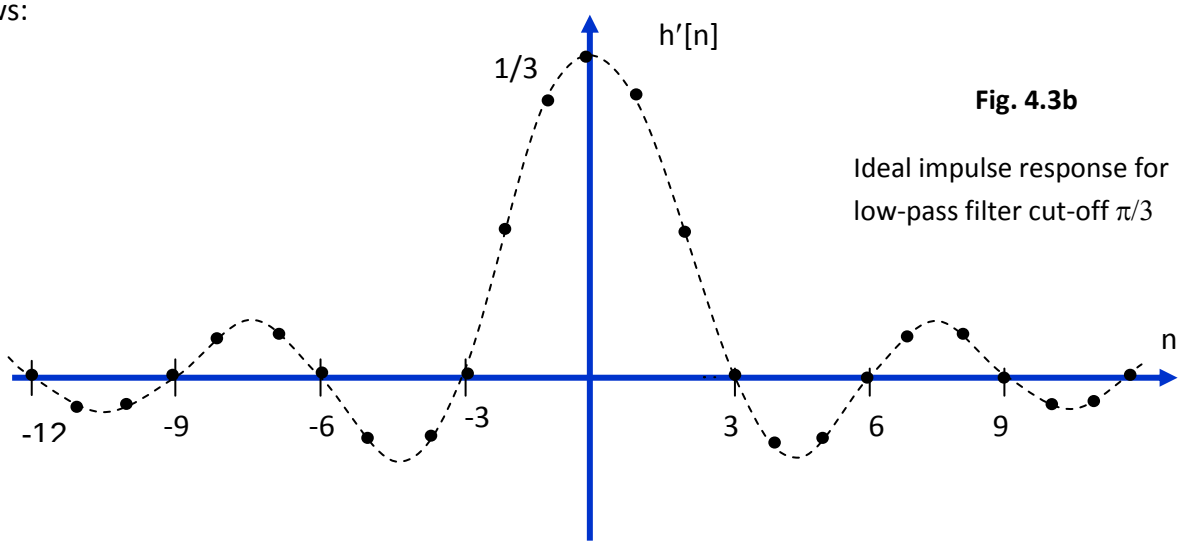


Fig. 4.3b

Ideal impulse response for low-pass filter cut-off $\pi/3$

In Fourier series method, limits of summation index is -∞ to ∞. But filter must have finite terms.

Hence limit of summation index change to -Q to Q where Q is some finite integer. But this type of truncation may result in poor convergence of the series. Abrupt truncation of infinite series is equivalent

to multiplying infinite series with rectangular sequence. i.e at the point of discontinuity some oscillation may be observed in resultant series.

2. Consider the example of LPF having desired frequency response Hd (ω) as shown

in figure. The oscillations or ringing takes place near band-edge of the filter.

3. This oscillation or ringing is generated because of side lobes in the frequency response

W(ω) of the window function. This oscillatory behavior is called "Gibbs Phenomenon".

Reading from the graph, or evaluating the formula, we get:

{h′[n]} = { ..., -0.055, -0.07, 0, 0.14, 0.28, <u>0.33</u>, 0.28, 0.14, 0, -0.07, -0.055, ... }

A digital filter with this impulse-response would have exactly the ideal frequency-response we applied to the inverse-DTFT i.e. a 'brick-wall' low-pass gain response & phase = 0 for all Ω. But

{h′[n]} has non-zero samples extending from n = -∞ to ∞, It is not a finite impulse-response. It is also not causal since h′[n] is not zero for all n < 0. It is therefore not realizable in practice.

To produce a realizable impulse-response of even order M:

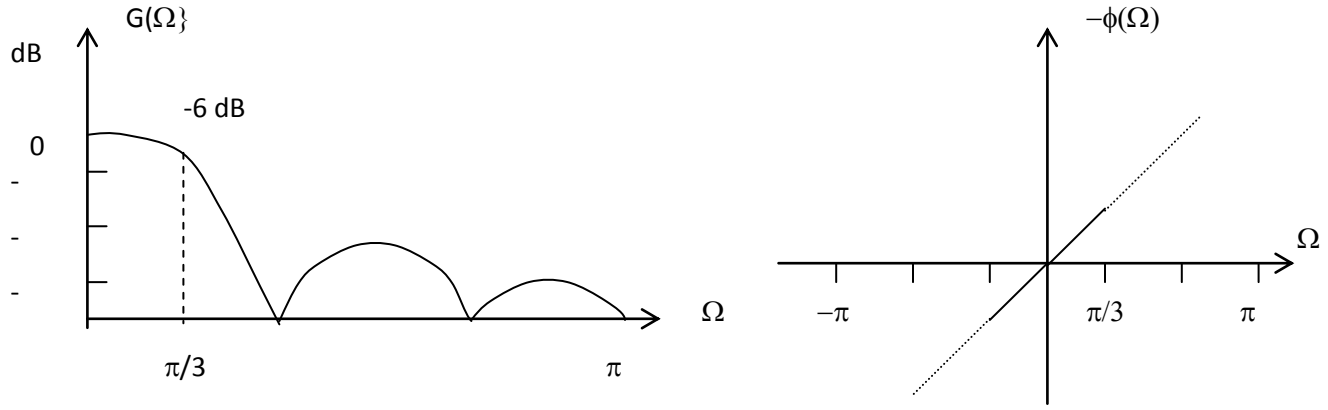
(1) Set $h[n] = \begin{cases} h'[n] & : \ \frac{-M}{2} \le n \le \frac{M}{2} \\ 0 & : \ \text{otherwise} \end{cases}$

(2) Delay resulting sequence by M/2 samples to ensure that the first non-zero sample occurs at n = 0.

The resulting causal impulse response may be realised by setting a_n = h[n] for n=0,1,2,...,M. Taking M=4, for example, the finite impulse response obtained for the π/3 cut-off low-pass specification is : {..,0,..,0, 0.14, 0.28, 0.33 , 0.28 , 0.14 , 0 ,..,0,..}

The resulting FIR filter is as shown in Figure 4.1 with a_0=0.14, a_1=0.28, a_2=0.33, a_3=0.28, a_4=0.14. (Note: a 4th order FIR filter has 4 delays & 5 multiplier coefficients).

The gain & phase responses of this FIR filter are sketched below.



Clearly, the effect of the truncation of {h[n]} to ±M/2 and the M/2 samples delay is to produce gain and phase responses which are different from those originally specified.

Considering the gain-response first, the cut-off rate is by no means sharp, and two 'ripples' appear in the stop-band, the peak of the first one being at about -21dB.

The phase-response is not zero for all values of Ω as was originally specified, but is linear phase (i.e. a straight line graph through the origin) in the pass-band of the low-pass filter (-π/3 to π /3) with slope arctan(M/2) with M = 4 in this case. This means that ϕ(Ω) = – (M/2)Ω for | Ω

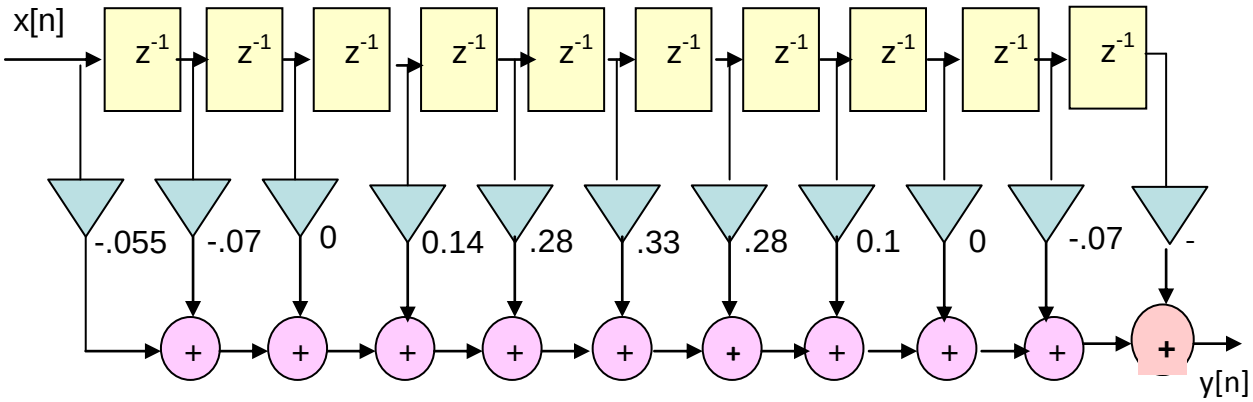
$| \leq \pi/3$; i.e. we get a linear phase-response (for $| \Omega | \leq \pi/3$) with a phase-delay of M/2 samples.

It may be shown that the phase-response is linear phase because the truncation was done symmetrically about n=0.

Now let's try to improve the low-pass filter by increasing the order to ten. Taking 11 terms of { (1 / 3) sinc (n / 3) } we get, after delaying by 5 samples:

{...0,-0.055,-.069, 0,.138,.276,.333,.276,.138,0,-.069,-.055,0,...}.

The signal-flow graph of the resulting 10^{th} order FIR filter is shown below:



Notice that the coefficients are again symmetric about the centre one (of value 0.33) and this again ensures that the FIR filter is linear phase.

([-0.055, -0.069, 0, 0.138, 0.276, 0.333, 0.276, 0.138, 0, -0.069, -0.055]);

In may be seen in the gain-response, as reproduced below, that the cut-off rate for the 10^{th} order FIR filter is sharper than for the 4th order case, there are more stop-band ripples and, rather disappointingly, the gain at the peak of the first ripple after the cut-off remains at about -21 dB. This effect is due to a well known property of Fourier series approximations, known as Gibb's phenomenon. The phase-response is linear phase in the pass band ($-\pi/3$ to $\pi/3$) with a phase delay of 5 samples. As seen in fig 4.6, going to 20th order produces even faster cut-off rates and more stop-band ripples, but the main stop-band ripple remains at about -21dB. This trend continues with 40^{th} and higher orders as may be easily verified. To improve matters we need to discuss '***windowing'.***

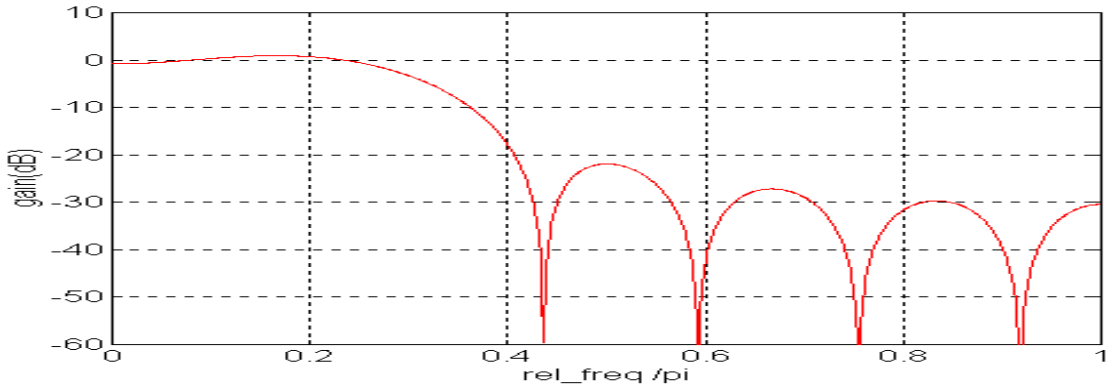


Fig : Gain response of tenth order low pass FIR filter with $\Omega_C = \pi/3$

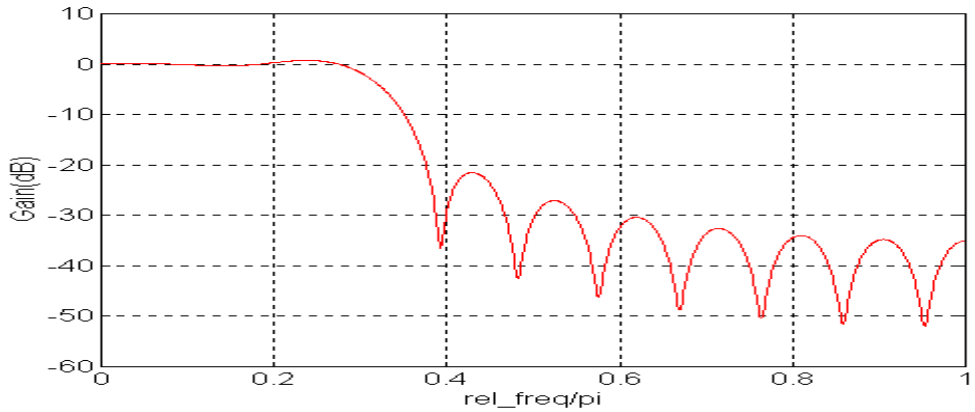


Fig : Gain response of 20th order low pass FIR filter with $\Omega_C = \pi/3$.

Windowing Technique:

FIR filter design using window functions

Windowing is the quickest method for designing an FIR filter. A windowing function simply truncates the ideal impulse response to obtain a causal FIR approximation that is non- causal an infinitely long. Smoother window functions provide higher out-of band rejection in the filter response. However this smoothness comes at the cost of wider stop band transitions. Various windowing method attempts to minimize the width of the main lobe (peak) of the frequency response. In addition, it attempts to minimize the side lobes (ripple) of the frequency response.

The FIR filter design process via window functions can be split into several steps:

- Defining filter specifications;
- Specifying a window function according to the filter specifications;

- Computing the filter order required for a given set of specifications;
- Computing the window function coefficients;
- Computing the ideal filter coefficients according to the filter order;
- Computing FIR filter coefficients according to the obtained window function and ideal filter coefficients;

If the resulting filter has too wide or too narrow transition region, it is necessary to change the filter order by increasing or decreasing it according to needs, and after that steps 4, 5 and 6 are iterated as many times as needed.

The final objective of defining filter specifications is to find the desired normalized frequencies (ωc, ωc1, ωc2), transition width and stop band attenuation. The window function and filter order are both specified according to these parameters. Accordingly, the selected window function must satisfy the given specifications. This point will be discussed in more detail in the next chapter .After this step, that is, when the window function is known, we can compute the filter order required for a given set of specifications. One of the techniques for computing is provided in chapter 2.3.When both the window function and filter order are known, it is possible to calculate the window function coefficients w[n] using the formula for the specified window function. This issue is also covered in the next chapter.After estimating the window function coefficients, it is necessary to find the ideal filter frequency samples. The expressions used for computing these samples are discussed in section 2.2.3 under Ideal filter approximation. The final objective of this step is to obtain the coefficients hd[n]. Two sequencies w[n] and hd[n] have the same number of elements. The next step is to compute the frequency response of designed filter h[n] using the following expression:

$$h[n] = w[n] \cdot h_d[n]$$

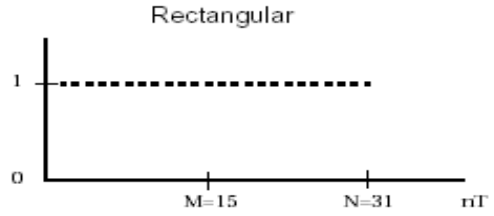
Lastly, the transfer function of designed filter will be found by transforming impulse response via Fourier transform:

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

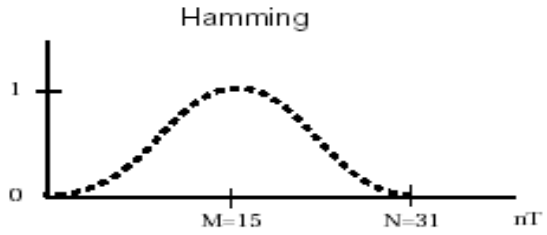
or via Z-transform $H(z) = \sum_{n=0}^{N} h[n] z^{-n}$ If the transition region of designed filter is wider than needed, it is necessary to increase the filter order, reestimate the window function coefficients and ideal filter frequency samples, multiply them in order to obtain the frequency response of designed filter and re estimate the transfer function as well. If the transition region is narrower than needed, the filter order can be decreased for the purpose of optimizing

hardware and/or software resources. It is also necessary to re estimate the filter frequency coefficients after that. For the sake of precise estimates, the filter order should be decreased or increased by 1.

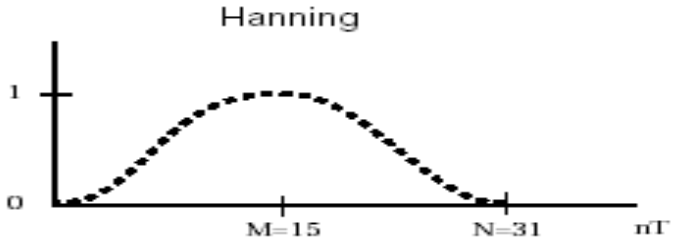
Rectangular Window: Rectangular This is the most basic of windowing methods. It does not require any operations because its values are either 1 or 0. It creates an abrupt discontinuity that results in sharp roll-offs but large ripples.



Hamming Window:This windowing method generates a moderately sharp central peak. Its ability to generate a maximally flat response makes it convenient for speech processing filtering.



Hanning Window: This windowing method generates a maximum flat filter design.



Kaiser Window: This windowing method is designed to generate a sharp central peak. It has reduced side lobes and transition band is also narrow. Thus commonly used in FIR filter design

- **Rectangular:**

$$w[n] = \begin{cases} 1 & 0 \le n \le N \\ 0 & \text{otherwise} \end{cases}$$

- **Bartlett (triangular):**

$$w[n] = \begin{cases} 2n/N & 0 \le n \le N/2 \\ 2 - 2n/N & N/2 < n \le N \\ 0 & \text{otherwise} \end{cases}$$

- **Hanning:**

$$w[n] = \begin{cases} 0.5 - 0.5\cos(2\pi n/N) & 0 \le n \le N \\ 0 & \text{otherwise} \end{cases}$$

- **Hamming:**

$$w[n] = \begin{cases} 0.54 - 0.46\cos(2\pi n/N) & 0 \le n \le N \\ 0 & \text{otherwise} \end{cases}$$

- **Kaiser:**

$$w[n] = \begin{cases} I_0[\beta(1 - [(n-\alpha)/\alpha]^2)^{1/2}] & 0 \le n \le N \\ 0 & \text{otherwise} \end{cases}$$

Name of window function $w(n)$	Mathematical definition
Rectangular	1
Hanning	$0.5 - 0.5\cos\left[\frac{2\pi n}{N-1}\right]$
Hamming	$0.54 - 0.46\cos\left[\frac{2\pi n}{N-1}\right]$
Blackman	$0.42 - 0.5\cos\left[\frac{2\pi n}{N-1}\right] + 0.08\cos\left[\frac{2\pi n}{N-1}\right]$

Type of window	Approx. Transition width of the main lobe	Peak Side lobe (dB)
Rectangular	4π/M	-13

Bartlett	8π/M	-27
Hanning	8π/M	-32
Hamming	8π/M	-43
Blackman	12π/M	-58

Looking at the above table we observe filters which are mathematically simple do not offer best characteristics. Among the window functions discussed Kaiser is the most complex one in terms of functional description whereas it is the one which offers maximum flexibility in the design.

Procedure for designing linear-phase FIR filters using windows:

1. Obtain $h_d(n)$ from the desired freq response using inverse FT relation
2. Truncate the infinite length of the impulse response to finite length with (assuming M odd) choosing proper window

$$h(n) = h_d(n)w(n) \text{ where}$$

$$w(n) \text{ is the window function defined for } -(M-1)/2 \le n \le (M-1)/2$$

3. Introduce h(n) = h(-n) for linear phase characteristics
4. Write the expression for H(z); this is non-causal realization
5. To obtain causal realization $H^{\cdot\cdot}(z) = z^{-(M-1)/2} H(z)$

Summary of 'windowing' design technique for FIR filters

To design an FIR digital filter of even order M, with gain response $G'(\Omega)$ and linear phase, by the windowing method,

1) Set $H'(e^{j\Omega}) = G'(\Omega)$ the required gain-response. This assumes $\phi'(\Omega) = 0$.
2) Inverse DTFT to produce the ideal impulse-response $\{h'[n]\}$.

3) Window to ±M/2 using chosen window.

4) Delay windowed impulse-response by M/2 samples.

5) Realize by setting multipliers of FIR filter.

Instead of obtaining $H'(e^{j\Omega}) = G'(\Omega)$, we get $e^{-j\Omega M/2}G(\Omega)$ with $G(\Omega)$ a distorted version of $G'(\Omega)$ the distortion being due to windowing.

The phase-response is therefore $\phi(\Omega) = -\Omega M/2$ which is a linear phase-response with phase-delay M/2 samples at all frequencies Ω in the range 0 to π. This is because $-\phi(\Omega) / \Omega = M/2$ for all Ω.

Notice that the filter coefficients, and hence the impulse-response of each of the digital filters we have designed so far are symmetric in that h[n] = h[M-n] for all n in the range 0 to M where M is the order. If M is even, this means that h[M/2 - n] = h[M/2 + n] for all n in the range 0 to M/2. The impulse response is then said to be 'symmetric' about sample M/2. The following example illustrates this for an example where M=6 and there are seven non-zero samples within {h[n]}: {… 0, …, 0, 2, -3, 5, 7, 5, -3, 2, 0, …,0, … }

The most usual case is where M is even, but, for completeness, we should briefly consider the case where M is odd. In this case, we can still say that {h[n]} is 'symmetric about M/2' even though sample M/2 does not exist. The following example illustrates the point for an example where M=5 and {h[n]} therefore has six non-zero sample:

(…, 0,…, 0, 1, 3, 5, 3, 1, 0, …, 0, …}

When M is odd, h[(M-1)/2 - n] = h[(M+1)/2 + n] for n = 0, 1, …, (M-1)/2.

It may be shown that FIR digital filters whose impulse-responses are symmetric in this way are linear phase. We can easily illustrate this for either of the two examples just given. Take the second. Its frequency-response is the DTFT of {h[n]} i.e.

It is also possible to design FIR filters which are not linear phase. The technique described in this section is known as the 'windowing' technique or the 'Fourier series approximation technique'.

FIR Filter Design Optimization

Remez Exchange Algorithm method:

An FIR digital filter design technique which is better than the windowing technique, but more complicated, is known as the 'Remez exchange algorithm'. It was developed by McClelland and Parks and is available in MATLAB. The following MATLAB program designs a 40^{th} order FIR low-pass filter whose gain is specified to be unity (i.e. 0 dB) in the range 0 to 0.3π radians/sample and zero in the range 0.4π to π. The gain in the " transition band " between 0.3π

and 0.4π is not specified. The 41 coefficients will be found in array 'a'. Notice that, in contrast to the gain-responses obtained from the 'windowing' technique, the Remez exchange algorithm produces 'equi-ripple' gain-responses (fig 4.14) where the peaks of the stop-band ripples are equal rather than decreasing with increasing frequency. The highest peak in the stop-band will be lower than that of an FIR filter of the same order designed by the windowing technique to have the same cut-off rate. Although they are a little difficult to see, there are 'equi-ripple' pass-band ripples.

a = remez (40, [0, 0.3, 0.4,1],[1, 1, 0, 0]);

freqz (a,1,1000,Fs);

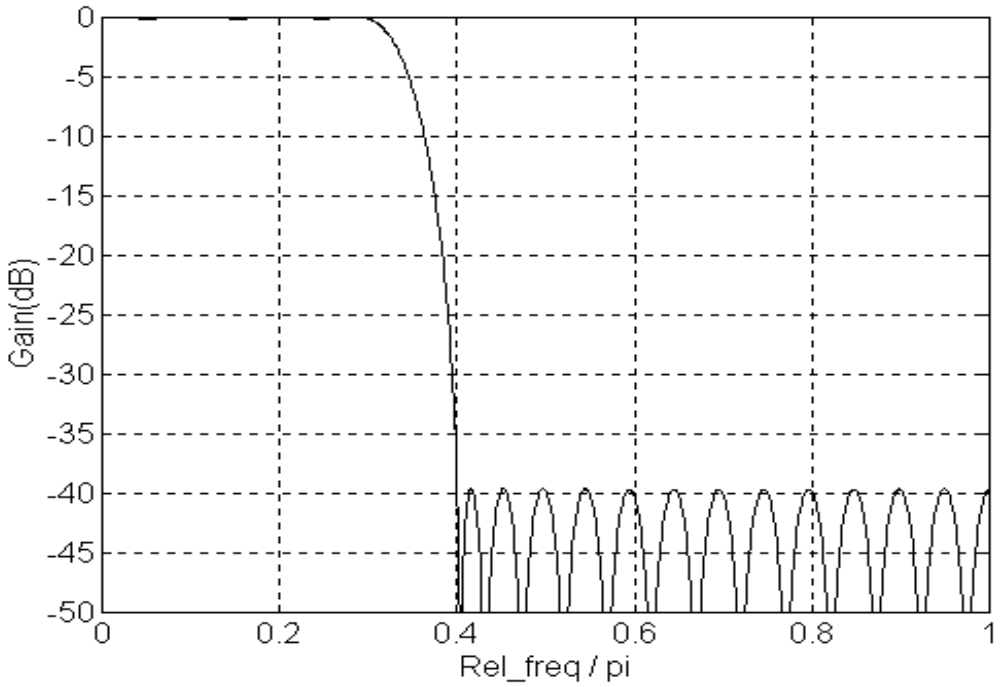


Fig 4.14: Gain response of 40th order FIR lowpass filter designed by " Remez "

These methods allow much greater flexibility in the filter specification. In general they seek the filter coefficients that minimize the error (in some sense) between a desired frequency response $H_d(e^{j\omega})$ and the achieved frequency response $H(e^{j\omega})$. The most common optimiza- tion method is that due to Parks and McClellan (1972) and is widely available in software filter design packages (including MATLAB)

The Parks-McClellan method allows

- Multiple pass- and stop-bands.

- Is an equi-ripple design in the pass- and stop-bands, but allows independent weighting of the ripple in each band.

- Allows specification of the band edges.

For the low-pass filter shown above the specification would be

$$1 - \delta_1 < H(e^{j\omega}) < 1 + \delta_1 \qquad \text{in the pass-band } 0 < \omega \leq \omega_c$$

$$-\delta_2 < H(e^{j\omega}) < \delta_2 \qquad \text{in the stop-band } \omega_s < \omega \leq \pi.$$

where the ripple amplitudes δ_1 and δ_2 need not be equal. Given these specifications we need to determine, the length of the filter $M + 1$ and the filter coefficients $\{h_n\}$ that meet the specifications in some optimal sense.

If $M + 1$ is odd, and we assume even symmetry

$$h_{M-k} = h_k \qquad k = 0 \ldots M/2$$

and the frequency response function can be written

$$\begin{aligned} H(e^{j\omega}) &= h_0 + 2\sum_{k=1}^{M/2} h_k \cos(\omega k) \\ &= \sum_{k=0}^{M/2} a_k \cos(\omega k) \end{aligned}$$

Let $H_d(e^{j\omega})$ be the desired frequency response, and define a weighted error

$$E(e^{j\omega}) = W(e^{j\omega})\, H_d(e^{j\omega}) - H(e^{j\omega})$$

where $W(e^{j\omega})$ is a frequency dependent weighting function, but by convention let $W(e^{j\omega})$ be constant across each of the critical bands, and zero in all transition bands. In particular for the low-pass design

$$W(e^{j\omega}) = \begin{cases} \delta_2/\delta_1 & \text{in the pass-band} \\ 1 & \text{in the stop-band} \\ 0 & \text{in the transition band.} \end{cases}$$

UNIT-V

APPLICATIONS OF DSP

Introduction

Multirate systems have gained popularity since the early 1980s and they are commonly used for audio and video processing, communications systems, and transform analysis to name but a few. In most applications multirate systems are used to improve the performance, or for increased computational efficiency. The two basic operations in a multirate system are decreasing (decimation) and increasing (interpolation) the sampling-rate of a signal. Multirate systems are sometimes used for sampling-rate conversion, which involves both decimation and interpolation.

Decimation

Decimation can be regarded as the discrete-time counterpart of sampling. Whereas in sampling we start with a continuous -time signal x(t) and convert it into a sequence of samples x[n], in decimation we start with a discrete-time signal x[n] and convert it into another discrete-time signal y[n], which consists of sub-samples of x[n]. Thus, the formal definition of M-fold decimation, or down -sampling, is defined. In decimation, the sampling rate is reduced from F_s to F_s/M by discarding M – 1 samples for every M samples in the original sequence.

An anti-aliasing digital filter precedes the down-sampler to prevent aliasing from occurring, due to the lower sampling rate. The subject of aliasing in decimated signals is covered in more detail in Section 9.4. In Figure 5.2 below, it illustrates the concept of 3-fold decimation i.e. M = 3. Here, the samples of x[n] corresponding to n = …, -2, 1, 4,… and n = …, -1, 2, 5,… are lost in the decimation process. In general, the samples of x[n] corresponding to $n \neq kM$, where k is an integer, are discarded in M-fold decimation. In Figure 5.2, it shows samples of the decimated signal y[n] spaced three times wider than the samples of x[n]. This is not a coincidence. In real time, the decimated signal appears at a slower rate than that of the original signal by a factor of M. If the sampling frequency of x[n] is F_s, then that of y[n] is F_s/M.

Interpolation

Interpolation is the exact opposite of decimation. It is an information preserving operation, in that all samples of $x[n]$ are present in the expanded signal $y[n]$. The mathematical definition of L-fold interpolation is defined by Equation 9.2 and the block diagram notation is depicted in Figure 9.3. Interpolation works by inserting $(L–1)$ zero-valued samples for each input sample. The sampling rate therefore increases from F_s to LF_s. With reference to Figure 5.3, the expansion process is followed by a unique digital low-pass filter called an *anti-imaging filter*. Although the expansion process does not cause aliasing in the interpolated signal, it does however yield undesirable replicas in the signal's frequency spectrum. We shall see how this special filter is necessary to remove these replicas from the frequency spectrum.

In Figure 5.4 below, it depicts 3-fold interpolation of the signal $x[n]$ i.e. $L = 3$. The insertion of zeros effectively attenuates the signal by L, so the output of the anti-imaging filter must be multiplied by L, to maintain the same signal magnitude.

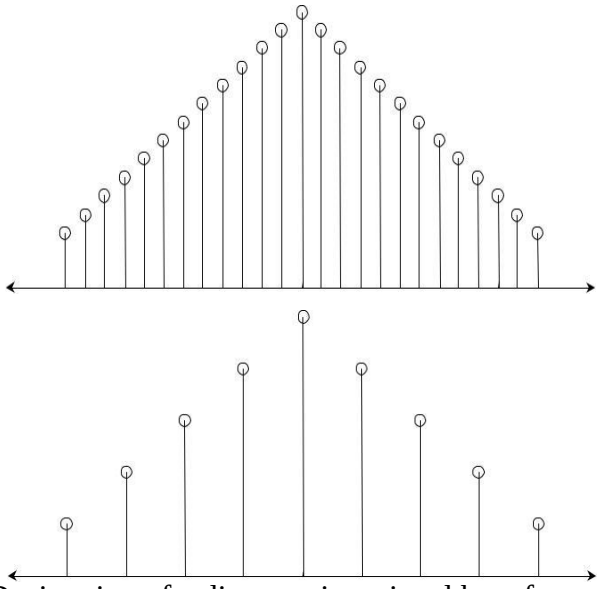


Figure :Decimation of a discrete-time signal by a factor of 3.

Frequency Transforms of Decimated and Expanded Sequences

The analysis of decimation and expansion is better understood by assessing their respective frequency spectrums using the Fourier transform.

Decimation

The implications of aliasing caused by decimation are very similar to those in the case of sampling a continuous-time signal, in above section. In general, if the Fourier transform of a signal, $X(\theta)$, occupies the entire bandwidth from $[-\pi, \pi]$, then the Fourier transform of the decimated signal, $X_{(\downarrow M)}(\theta)$, will be aliased. This is due to the superposition of the M shifted and frequency-scaled transforms. This is illustrated in Figure 5.5 below, which shows the aliasing phenomenon for $M = 3$.

.

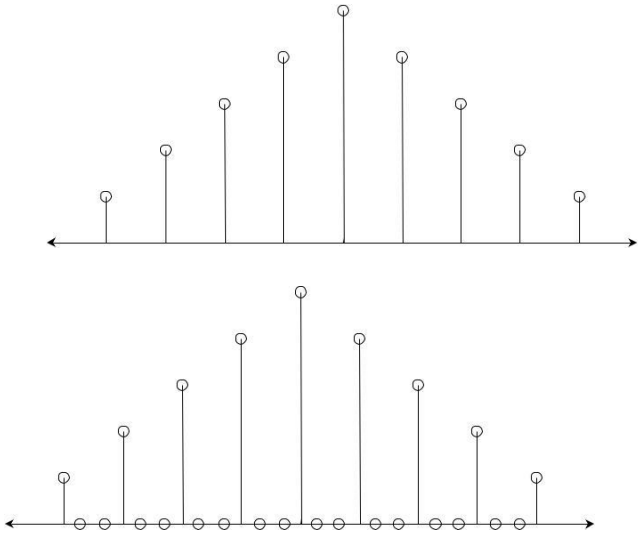
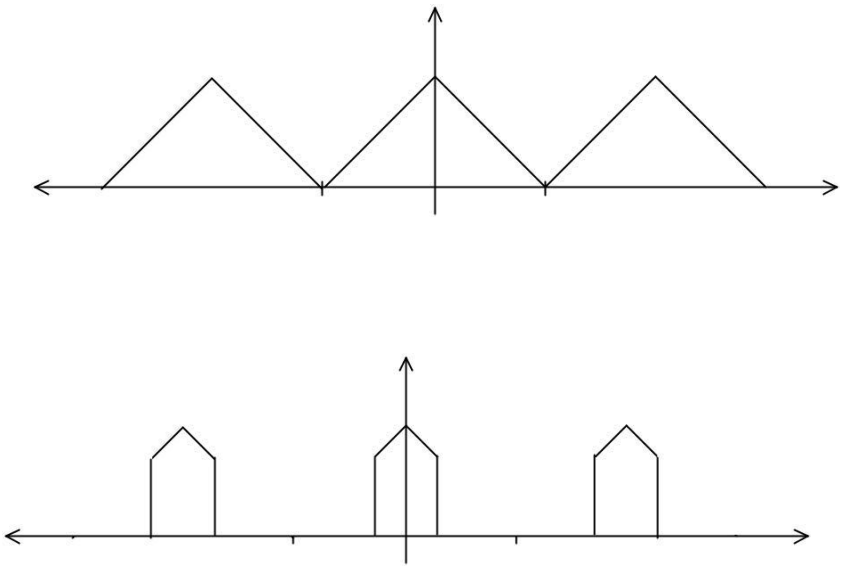


Figure 5.2: Interpolation of a discrete-time signal by a factor of 3



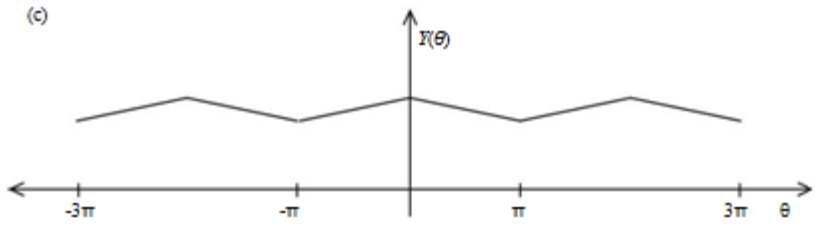


Figure 5.3: Aliasing caused by decimation; (a) Fourier transform of the original signal; (b) After decimation filtering; (c) Fourier transform of the decimated signal.

In Figure 5.5 it shows the Fourier transform of the original signal. Part (b) shows the signal after lowpass filtering.

In Figure 5.5, it depicts the expanded spectrum after decimation.

Expansion

The effect of expansion on a signal in the frequency domain is illustrated in Figure 5.6 below. Part (a) shows the Fourier transform of the original signal; part (b) illustrates the Fourier transform of the signal with zeros added $W(\theta)$; and part shows the Fourier transform of the signal after the interpolation filter.

It is clearly visible that the shape of the Fourier transform is compressed by a factor L in the frequency axis and is also repeated L times in the range of $[-\pi, \pi]$. Despite the compression of the signal in the frequency axis, the shape of the Fourier transform is still preserved, confirming that expansion does not lead to aliasing.

These replicas are removed by a digital low-pass filter called an *anti-imaging* filter, as indicated in Figure 5.3.

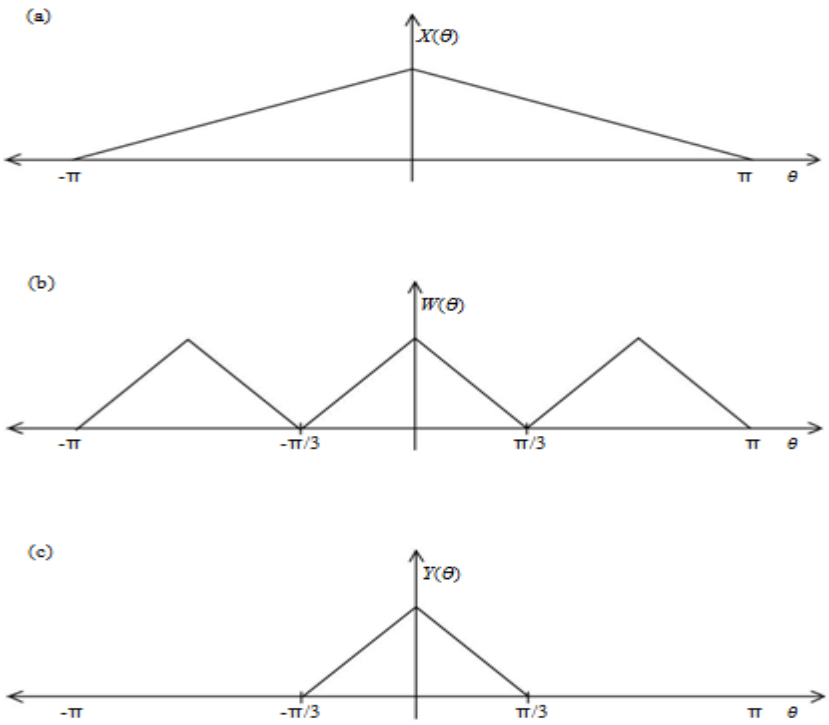


Figure 5.4: Expansion in the frequency domain of the original signal (a) and the expanded signal

Sampling-rate Conversion

A common use of multirate signal processing is for sampling-rate conversion. Suppose a digital signal $x[n]$ is sampled at an interval T_1, and we wish to obtain a signal $y[n]$ sampled at an interval T_2. Then the techniques of decimation and interpolation enable this operation, providing the ratio T_1/T_2 is a rational number i.e. L/M.

Sampling-rate conversion can be accomplished by L -fold expansion, followed by low-pass filtering and then M- fold decimation, as depicted in Figure 5.7. It is important to emphasis that the interpolation should be performed first and decimation second, to preserve the desired spectral characteristics of $x[n]$. Furthermore by cascading the two in this manner, both of the filters can be combined into one single low-pass filter.

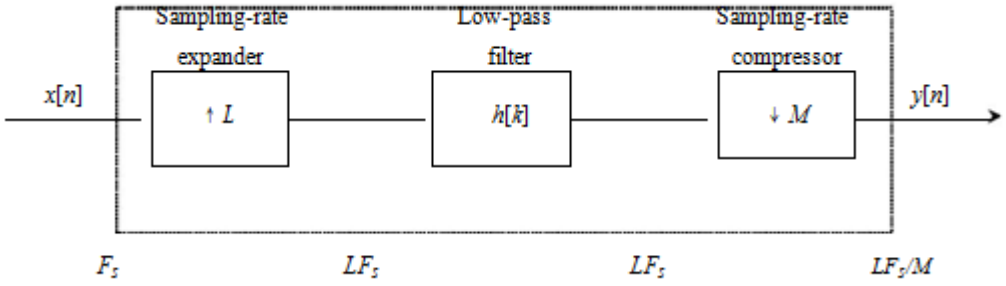


Figure 5.5: Sampling-rate conversion by expansion, filtering, and decimation.

An example of sampling-rate conversion would take place when data from a CD is transferred onto a DAT. Here the sampling-rate is increased from 44.1 kHz to 48 kHz. To enable this process the non-integer factor has to be approximated by a rational number: Hence, the sampling-rate conversion is achieved by interpolating by L i.e. from 44.1 kHz to [44.1x160] = 7056 kHz.

Then decimating by M i.e. from 7056 kHz to [7056/147] = 48 kHz.

Multistage Approach

When the sampling-rate changes are large, it is often better to perform the operation in multiple stages, where $M_i(L_i)$, an integer, is the factor for the stage i.

$$M = M_1 M_2 \ldots M_I \quad \text{or} \quad L = L_1 L_2 \ldots L_I$$

An example of the multistage approach for decimation is shown in Figure 9.8. The multistage approach allows a significant relaxation of the anti-alias and anti-imaging filters, with a consequent reduction in the filter complexity. The optimum number of stages is one that leads to the least computational effort in terms of either the multiplications per second (MPS), or the total storage requirement (TSR).

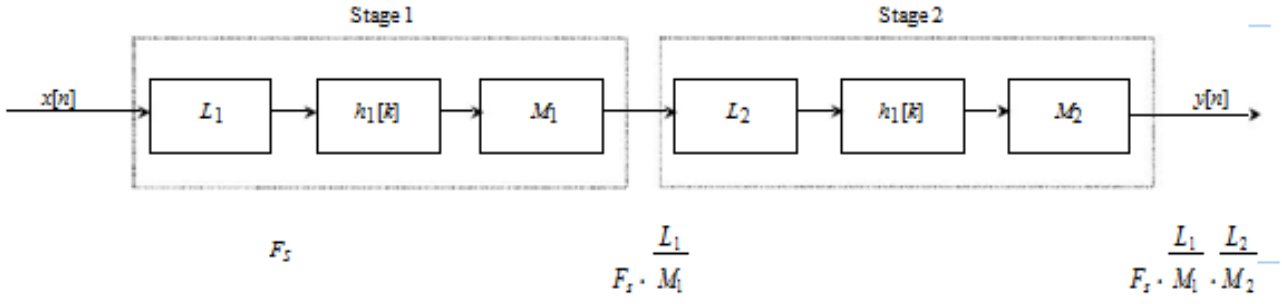


Figure 5.6: Multistage approach for the decimation process.

Polyphase Filters

Potential computational savings can be made within the process of decimation, interpolation, and sampling-rate conversion. *Polyphase filters* is the name given to certain realisations of multirate filtering operations, which facilitate computational savings in both hardware and software. As an example, the combined low-pass filter in the sampling-rate converter, as illustrated in Figure 5.7, can be re-drawn as a realisation structure. In principle, the simplest realisation of the low-pass filter is the direct-form FIR structure, as depicted in Figure 5.9. However, this type of structure is very inefficient owing to the interpolation process, which introduces (L–1) zeros between consecutive points in the signal. If L is large, then the majority of the signal components fed into the FIR filter are zero. As a result, most of the multiplications and additions are zero i.e. many pointless calculations. Furthermore, the decimation process itself implies that only one out of every M output samples is required at the output of the sampling-rate converter. Consequently, only one out of every M possible values at the output of the filter needs to be computed. This type of structure therefore, leads to much inefficiency during the process of sampling-rate conversion. A more efficient realisation structure of the sampling-rate converter uses polyphase filters, as illustrated in Figure 5.10. It takes into account that after the interpolation process the signal consists of (L–1) zero coefficients, and the decimation process implies that only one out of every M samples is required at the output of the converter. To make the scheme more efficient, the low-pass filter in Figure 5.9 is replaced by a *bank of filters* arranged in *parallel*, as illustrated in the efficient realisation. The sampling-rate conversion process is undertaken by the multiplexer at the output by selecting every MT/L samples. In this example, the efficient realisation is illustrated for a signal which is interpolated by L = 3 and decimated by M = 2 samples.

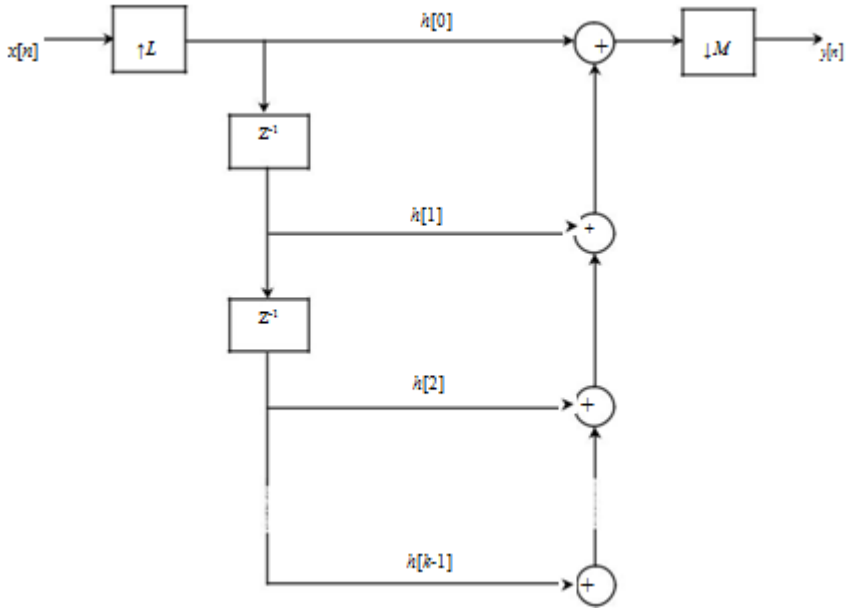


Figure :Realization structure of sampling-rate conversion.

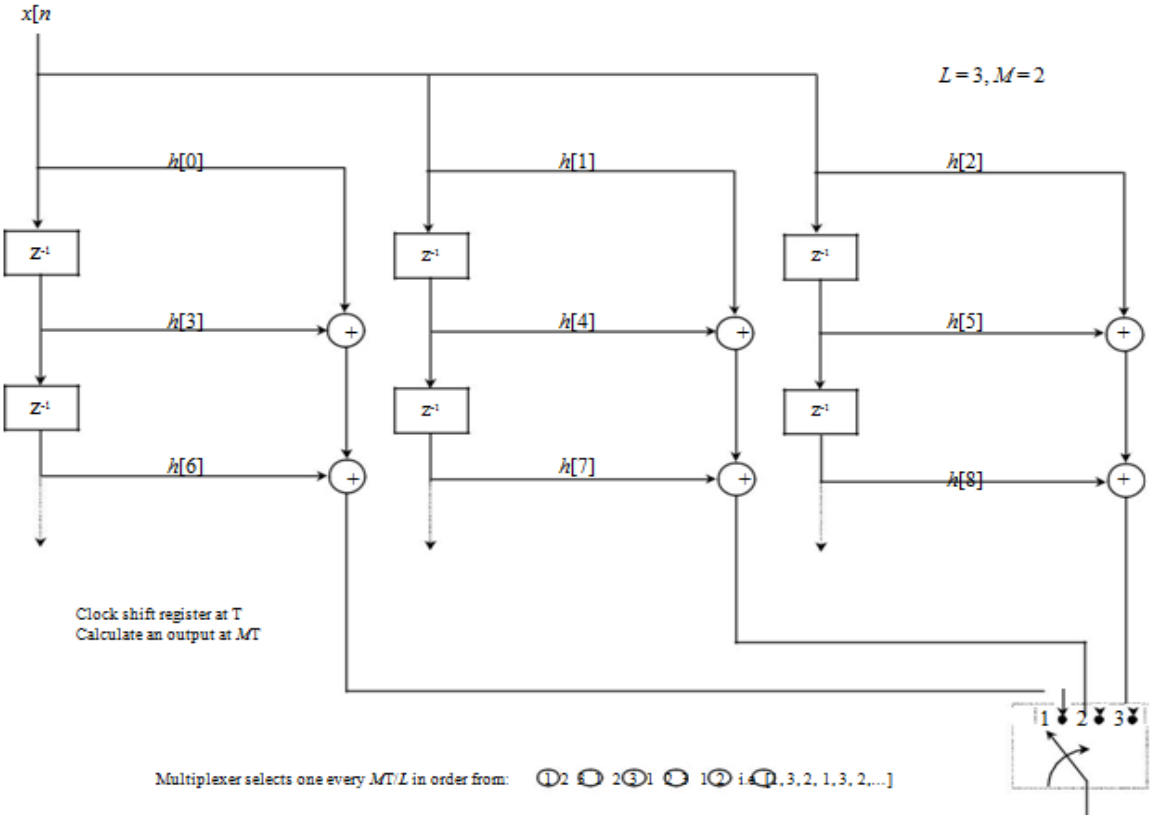


Figure: Efficient realization structure for sampling-rate conversion.

Applications of Multirate DSP

Multirate systems are used in a CD player when the music signal is converted from digital into analogue (DAC). Digital data (16-bit words) are read from the disk at a sampling rate of 44.1 kHz. If this data were converted directly into an analogue signal, image frequency bands centred on multiples of the sampling-rate would occur, causing amplifier overload, and distortion in the music signal. To protect against this, a common technique called *oversampling* is often implemented nowadays in all CD players and in most digital processing systems of music signals. Figure 5.9 below illustrates a basic block diagram of a CD player and how oversampling is utilised. It is customary to oversample (or expand) the digital signal by a factor of x8, followed by an interpolation filter to remove the image frequencies. The sampling rate of the resulting signal is now increased up to 352.8 kHz. The digital signal is then converted into an analogue waveform by passing it through a 14-bit DAC. Then the output from this device is passed through an analogue low-pass filter before it is sent to the speakers.

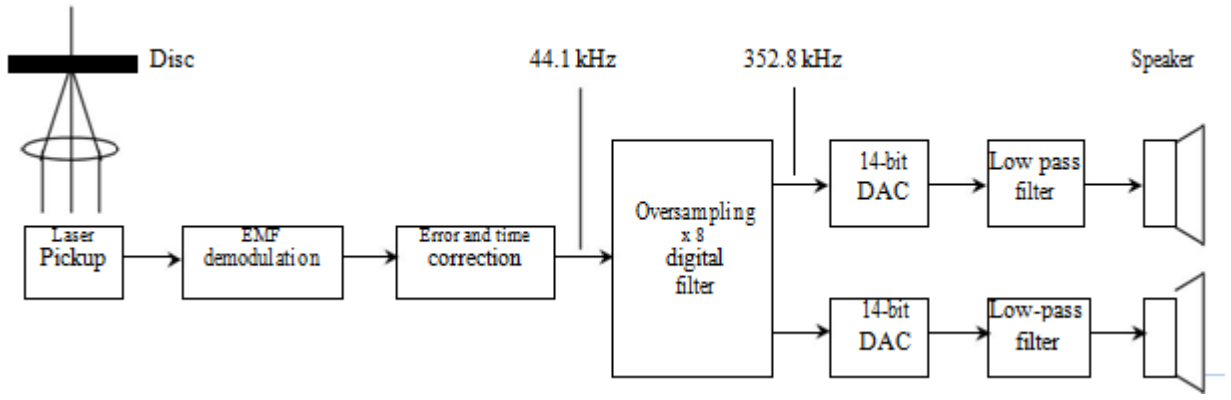


Figure: Digital to analogue conversion for a CD player using x8 oversampling.

Above Figure illustrates the procedure of converting a digital waveform into an analogue signal in a CD player using x8 oversampling. As an example, Figure (a) illustrates a 20 kHz sinusoidal signal sampled at 44.1 kHz, denoted by $x[\ n]$. The six samples of the signal represent the waveform over two periods. If the signal $x[n]$ was converted directly into an analogue waveform, it would be very hard to exactly reconstruct the 20 kHz signal from this diagram. Now, Figure (b) shows $x[n]$ with an x8 interpolation, denoted by $y[n]$. Figure (c) shows the analogue signal $y(t)$, reconstructed from the digital signal $y[n]$ by passing it through a DAC. Finally, Figure (d) shows the waveform of $z(t)$, which is obtained by passing the signal $y(t)$ through an analogue low-pass filter.

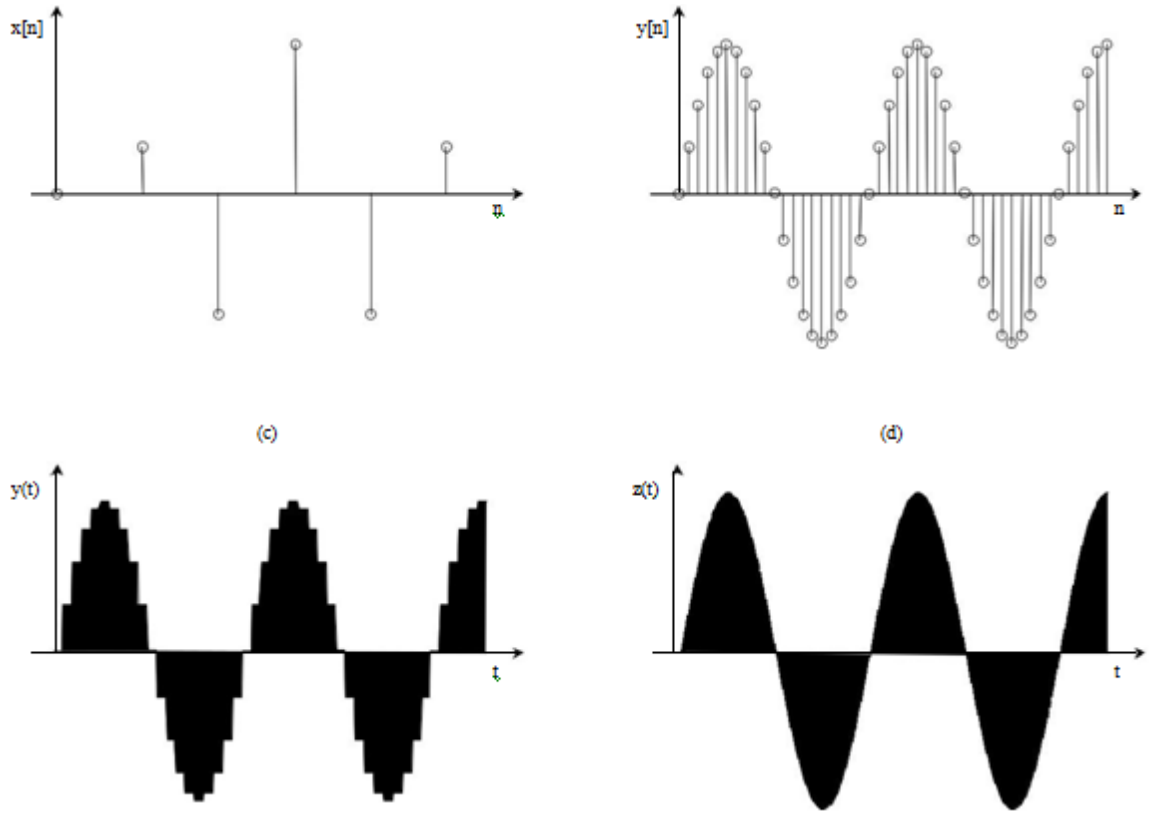


Figure :Illustration of oversampling in CD music signal reconstruction.

The effect of oversampling also has some other desirable features. Firstly, it causes the image frequencies to be much higher and therefore easier to filter out. The anti-alias filter specification can therefore be very much relaxed i.e. the cut-off frequency of the filter for the previous example increases from [44.1 / 2] = 22.05 kHz to [44.1x8 / 2] = 176.4 kHz after the interpolation.

One other attractive feature about oversampling is the effect of reducing the *noise power spectral density*, by spreading the noise power over a larger bandwidth. This is illustrated in Figure 9.13 and mathematical defined.

Noise power spectral density= Total power /Bandwidth

For both sequences, the *total noise power* (shaded area in Figure 5.8) remains the same. However, as the bandwidth is increased by a factor of x8 because of the interpolation process, it

causes the level of the noise power spectral density to decrease by a factor of x8, over the whole range of the bandwidth.

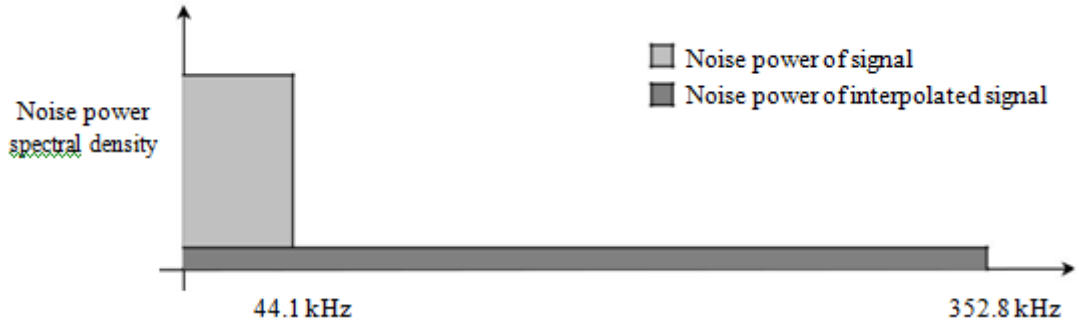


Figure :Illustration of noise power spectral density reduction due to oversampling.

As a consequence of the reduction in the noise power spectral density, it means that the level of tolerable noise can be increased by a factor of 8. In terms of the quantisation noise power, q^2, it means that it can now be 8 times greater (or the quantisation step size, q, can be increased by $\sqrt{8}$). This ultimately means that a reduction in the number of bits for the DAC is possible. In general, the reduction in the number of bits for the DAC process is given by Equation 9.4 below.

DAC bit reduction=1/2 $\log_2$ (*oversample factor*)

For the previous example, the DAC bit reduction owing to the x8 oversample factor is $1/2\log_2(8) = 1.5$ bits.

There are in fact more sophisticated oversampled ADCs and DACs that use various feedback paths within the system to move most of the quantisation noise into a high frequency out-of-band region. Substantially larger savings in the number of bits can then be made, even to one bit only, but these techniques are beyond the topic of this course.

Finite Word-length Effects

Practical digital filters must be implemented with finite precision numbers and arithmetic. As a result, both the filter coefficients and the filter input and output signals are in discrete form. This leads to four types of finite wordlength effects. Discretization (quantization) of the filter coefficients has the effect of perturbing the location of the filter poles and zeroes. As a result, the actual filter response differs slightly from the ideal response. This *deterministic* frequency response error is referred to as **coefficient**

quantization error. The use of finite precision arithmetic makes it necessary to quantize filter calculations by rounding or truncation. **Roundoff noise** is that error in the filter output that results from rounding or truncating calculations within the filter. As the name implies, this error looks like low-level noise at the filter output. Quantization of the filter calculations also renders the filter slightly nonlinear. For large signals this nonlinearity is negligible and roundoff noise is the major concern. However, for recursive filters with a zero or constant input, this nonlinearity can cause spurious oscillations called **limit cycles**. With fixed-point arithmetic it is possible for filter calculations to overflow. The term **overflow oscillation**, sometimes also called **adder overflow limit cycle**, refers to a high-level oscillation that can exist in an otherwise stable filter due to the nonlinearity associated with the overflow of internal filter calculations. In this chapter, we examine each of these finite wordlength effects. Both fixed-point and floating-point number representations are considered.

Number Representation

In digital signal processing, (B C 1)-bit fixed-point numbers are usually represented as two's-complement signed fractions in the format

$$b_0 \cdot b_{-1}b_{-2} \cdots b_{-B}$$

The number represented is then

$$X = -b_0 + b_{-1}2^{-1} + b_{-2}2^{-2} + \cdots + b_{-B}2^{-B}$$

where b_0 is the sign bit and the number range is -1 X < 1. The advantage of this representation is that the product of two numbers in the range from −1 to 1 is another number in the same range.

Floating-point numbers are represented as

$$X = (-1)^s m 2^c$$

where *s* is the *sign bit*, *m* is the **mantissa**, and *c* is the *characteristic* or *exponent*. To make the representation of a number unique, the mantissa is *normalized* so that 0:5 *m* < 1.

Although floating-point numbers are always represented in the form of (3.2), the way in which this representation is actually *stored* in a machine may differ. Since *m* 0:5, it is not necessary to store the 2^{-1}-weight bit of *m*, which is always set. Therefore, in practice numbers are usually stored as

$$X = (-1)^s(0.5 + f)2^c$$

where *f* is an unsigned fraction, 0 *f* < 0:5.

Most floating-point processors now use the IEEE Standard 754 32-bit floating-point format for storing numbers. According to this standard the exponent is stored as an unsigned integer *p* where

$$p = c + 126$$

Therefore, a number is stored as

$$X = (-1)^s(0.5 + f)2^{p-126}$$

where *s* is the sign bit, *f* is a 23-b unsigned fraction in the range 0 *f* < 0:5, and *p* is an 8-b

unsigned integer in the range 0 *p* 255. The total number of bits is 1 C 23 C 8 D 32. For example, in IEEE format 3=4 is written .−1/^{0}.0:5 C 0:25/2^0 so *s* D 0, *p* D 126, and *f* D 0:25.

The value *X* D 0 is a unique case and is represented by all bits zero (i.e., *s* D 0, *f* D 0, and *p* D 0). Although the 2^{-1}-weight mantissa bit is not actually stored, it does exist so the mantissa has 24 b plus a sign bit.

Fixed-Point Quantization Errors

In fixed-point arithmetic, a multiply doubles the number of significant bits. For example, the product of the two 5-b numbers 0.0011 and 0.1001 is the 10-b number 00:000 110 11. The extra bit to the left of the decimal point can be discarded without introducing any error. However, the least significant four of the remaining bits must ultimately be discarded by some form of quantization so that the result can be stored to 5 b for use in other calculations. In the example above this results in 0.0010 (quantization by rounding) or 0.0001 (quantization by truncating). When a sum of products calculation is performed, the quantization can be performed either after each multiply or after all products have been summed with double-length precision.

We will examine three types of fixed-point quantization—rounding, truncation, and magnitude truncation. If X is an exact value, then the rounded value will be denoted Q_r $.X/$, the truncated value Q_t $.X/$, and the magnitude truncated value Q_{mt} $.X/$. If the quantized value has B bits to the right of the decimal point, the quantization step size is

$$\Delta = 2^{-B} \qquad)$$

Since rounding selects the quantized value nearest the unquantized value, it gives a value which is never more than $1=2$ away from the exact value.

The error resulting from quantization can be modeled as a random variable uniformly distributed over the appropriate error range. Therefore, calculations with round off error can be considered error-free calculations that have been corrupted by additive white noise. The mean of this noise for rounding is where Efg represents the operation of taking the expected value of a random variable.

Floating-Point Quantization Errors

With floating-point arithmetic it is necessary to quantize after both multiplications and additions. The addition quantization arises because, prior to addition, the mantissa of the smaller number in the sum is shifted right until the exponent of both numbers is the same. In general, this gives a sum mantissa that is too long and so must be quantized. We will assume that quantization in floating-point arithmetic is performed by rounding. Because of the exponent in floating-point arithmetic, it is the relative error that is important. The relative error is defined as

$$\varepsilon_r = \frac{Q_r(X) - X}{X} = \frac{\epsilon_r}{X}$$

Since $X = (-1)^s m 2^c$, $Q_r(X) = (-1)^s Q_r(m) 2^c$ and

$$\varepsilon_r = \frac{Q_r(m) - m}{m} = \frac{\epsilon}{m}$$

Roundoff Noise

To determine the roundoff noise at the output of a digital filter we will assume that the noise due to a quantization is stationary, white, and uncorrelated with the filter input, output, and internal variables. This assumption is good if the filter input changes from sample to sample in a sufficiently complex manner. It is not valid for zero or constant inputs for which the effects of rounding are analyzed from a limit cycle perspective.

To satisfy the assumption of a sufficiently complex input, roundoff noise in digital filters is often calculated for the case of a zero-mean white noise filter input signal *x.n/* of variance $_x^2$. This simplifies calculation of the output roundoff noise because expected values of the form *Efx.n/x.n − k/*g are zero for *k* 6D0 and give $_x^2$ when *k* D 0. This approach to analysis has been found to give estimates of the output roundoff noise that are close to the noise actually observed for other input signals. Another assumption that will be made in calculating roundoff noise is that the product of two quantization errors is zero. To justify this assumption, consider the case of a 16-b fixed-point processor. In this case a quantization error is of the order 2^{-15}, while the product of two quantization errors is of the order 2^{-30}, which is negligible by comparison. If a linear system with impulse response *g.n/* is excited by white noise with mean m_x and variance $_x^2$, the output is noise of mean

$$m_y = m_x \sum_{n=-\infty}^{\infty} g(n)$$

and variance

$$\sigma_y^2 = \sigma_x^2 \sum_{n=-\infty}^{\infty} g^2(n)$$

Therefore, if *g.n/* is the impulse response from the point where a roundoff takes place to the filter output, the contribution of that roundoff to the variance (mean-square value) of the output roundoff noise is given by (3.25) with $_x^2$ replaced with the variance of the roundoff. If there is more than one source of roundoff error in the filter, it is assumed that the errors are uncorrelated so the output noise variance is simply the sum of the contributions from each source.

Roundoff Noise in FIR Filters

The simplest case to analyze is a finite impulse response (FIR) filter realized via the convolution summation

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

When fixed-point arithmetic is used and quantization is performed after each multiply, the result of the *N* multiplies is *N* -times the quantization noise of a single multiply. For example, rounding after each multiply gives, from (3.6) and (3.12), an output noise variance of

$$\sigma_o^2 = N\frac{2^{-2B}}{12}$$

Limit Cycles

A limit cycle, sometimes referred to as a multiplier round-off limit cycle, is a low-level oscillation that can exist in an otherwise stable filter as a result of the nonlinearity associated with rounding (or truncating) internal filter calculations. Limit cycles require recursion to exist and do not occur in non-recursive FIR filters.

Overflow Oscillations

With fixed-point arithmetic it is possible for filter calculations to overflow. This happens when two numbers of the same sign add to give a value having magnitude greater than one. Since numbers with magnitude greater than one are not representable, the result

overflows. For example, the two's complement numbers 0.101 (5/8) and 0.100 (4/8) add to give 1.001 which is the two's complement representation of $-7=8$.

The overflow characteristic of two's complement arithmetic can be represented as $R\{\}$ where

$$R\{X\} = \begin{cases} X - 2 & X \geq 1 \\ X & -1 \leq X < 1 \\ X + 2 & X < -1 \end{cases}$$