Spring 2021

EE433 Real-time Applications of Digital Signal Processing

Definitions of Hardware-Software Components of Embedded Systems

In this part, definitions of frequently used terms in the context of embedded systems are given.

Real Time Processing:

Real-time operation or processing guarantees the response within a specified time constraint or event deadline. In real-time processing, a specified deadline relative to an event must always be met regardless of system load in a deterministic manner.

GPOS: General purpose operating system (Linux, Windows, Android, IoS, etc.)

GPOS has scheduler which decides to run programs in a sequence rapidly switching between them based on the program priority. This results in the illusion of simultaneous program execution. GPOS does not satisfy real-time processing constraints and hence they are not real-time operating systems (RTOS).

RTOS: Real-time operating system (RT Linux, Lynx, Wind River Linux, QNX)

RTOS provides deterministic execution pattern and time. It is employed to ensure that a process is executed in a given timeframe. RTOS is light weight and does not include functions which exist in GPOS. It is especially employed in mission critical applications such as airplanes, and automobiles.

CPU: Central processing unit

CPU is the core component of a computing device. In its simplest form, CPU takes instructions from a program or application and performs a calculation. Modern CPUs have multiple cores or processing units. A CPU with two cores, for example, could run two different processes at the same time. This speeds up your system, because your computer can do multiple things at once.

GPU: Graphical processing unit

GPU has a highly parallel structure making it more efficient and effective than CPU for processing large blocks of data in parallel. GPU can be present on a video card or embedded on the motherboard. GPUs are especially useful for machine learning, video editing, and gaming applications.

FPGA: Field programmable gate array

FPGA is a reconfigurable hardware composed of programmable logic blocks to implement complex digital computations. FPGA can be programmed by VHDL of Verilog design languages. The programming complexity in FPGA is high compared to CPU but significant advances are made in software for flexibility.

ASIC: Application specific integrated circuit

ASIC is an integrated circuit (IC) customized for a specific function. ASIC cannot be reprogrammed after manufacturing as opposed to FPGA. The advantage of ASIC is faster computation of a specific task as well

as its cost. ASICs are manufactured in 10000s quantities and they are used for final series production of devices.

SOM: System on module

SOM is a board that is composed of several components to implement a system function. Usually it includes USB, Ethernet, and HDMI ports in addition to CPU, FPGA, GPU, RAM and more.

SOC: System on chip

SOC is an integrated circuit (IC) that has most of the components of a computer or electronic system. These components may include CPU, GPU, FPGA and RAM.

Some Popular Programming Languages:

MATLAB:

MATLAB is the most user friendly, easy to learn, program and debug programming language for engineers. It is a must know language with a very wide set of libraries and functions. Most of the time, a proof of concept algorithm is first coded in MATLAB and then converted to another language for better speed and compatibility with the hardware. It is at the top of the high level languages and it is an interpreter which executes command line by line instead of a compilation process.

Python:

Python is one of the most preferred language after MATLAB since it presents a large number of libraries and built-in functions similar to MATLAB. Python is especially the language of choice for machine learning. It is also an interpreter and hence speed is the main disadvantage compare to compiled languages.

Java and JavaScript:

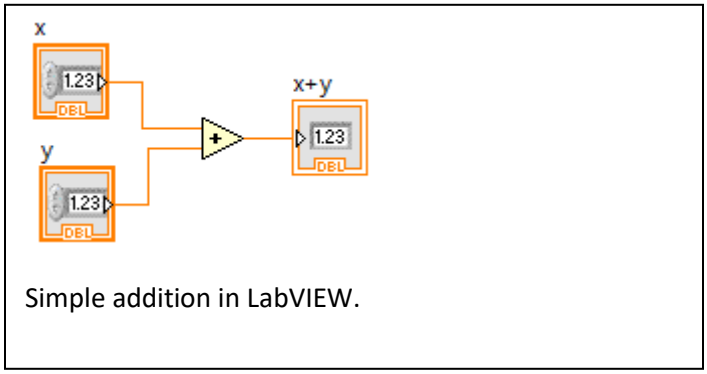
JavaScript is an interpreter and each command is executed in line. Java code must be compiled and then executed. Java is an object-oriented language. JavaScript is an object-oriented scripting language. Java is used in servers while JavaScript is used in client-side for validation, and web page development.

C:

C is a general-purpose programming language which is frequently used in embedded systems. It has a compiler with optimization options. C is the industry standard for implementations where the speed is the ultimate factor.

LabVIEW:

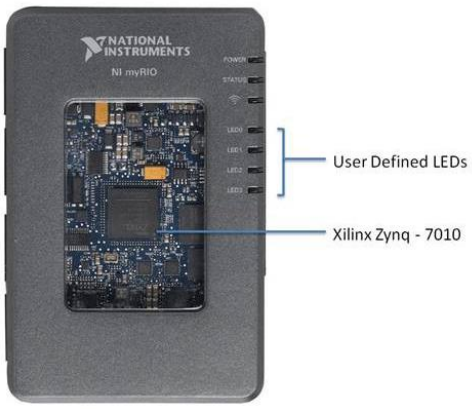
LabVIEW is a system-design platform and a visual programming language from National Instruments (NI). It is similar to Simulink but it is more advanced in the sense that it can be used to control several hardware units including, CPU, FPGA, USB, Ethernet, etc. Each function in LabVIEW has a compiled C code running on the background. NI has several boards for real-time embedded system programming. One simple example is the myRIO. In LabVIEW, programming is done through symbolic blocks for each operation. You can write your code as the interconnection of these blocks and create your user interface to observe the program outcomes. A simple add operation is shown below.

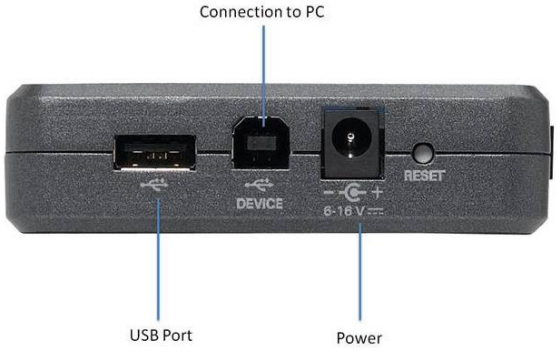


Simple addition in LabVIEW.

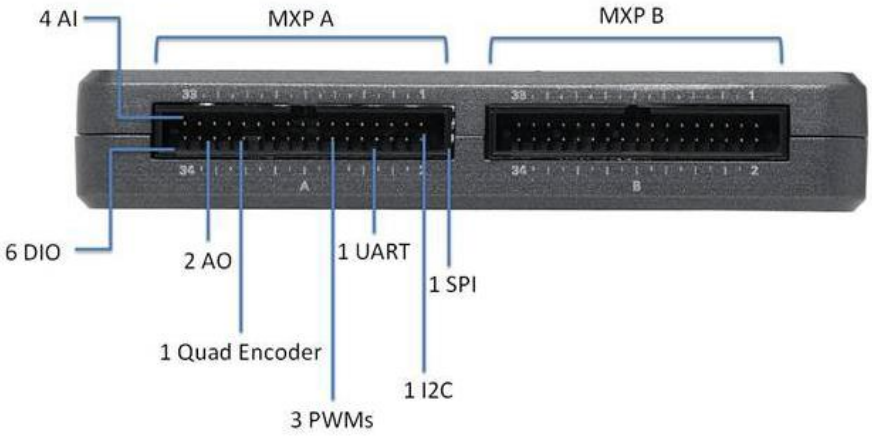
myRIO:

myRIO is an entry level SOM which has CPU, FPGA, A/D, D/A, USB, WiFi, Ethernet, audio in/out, three-axis accelerometer and more. myRIO includes Xilinx Zynq-7010 SoC which is an example of a dual-core ARM® Cortex™-A9 processor and an FPGA with 28,000 programmable logic cells, 10 analog inputs, 6 analog outputs, audio I/O channels, and up to 40 lines of digital input/output (DIO). All the input and output ports are easily accessible. It is a low-cost embedded system but it has all the functionalities of mode advanced systems with limited computational power. myRIO requires LabVIEW for programming tasks. myRIO has a compact form and some of the connectors on board are shown in the following figure.





Identical Connectors



EE433

Real-time Applications of Digital Signal Processing

EE433 focuses on the key applications of Digital Signal Processing in embedded systems. The embedded system currently used is NI myRIO and the programming tool is LabVIEW. EE433l considers the applications in CPU and FPGA by taking inputs from the analog ports of myRIO and applying them to the built-in A/D converter. Similarly digital signals are converted to analog by using D/A and the analog outputs are driven from the analog output ports of myRIO. In this part, we will refresh our background on theoretical concepts of Sampling in order to comfortably apply them in out practical development environment.

1. Sampling of Continuous-time Signals

Continuous-time signals should be converted to digital form in order to process them in digital systems. Theoretically the sampling operation is performed through the ideal sampling block given below.

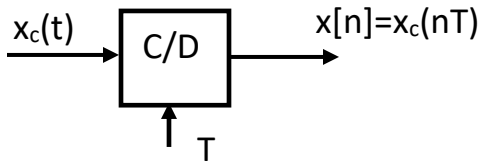


Fig1. Ideal continuous-to-discrete time converter.

In practice, C/D converter is implemented by using analog-to-digital converter, or A/D converter. A/D converter is usually a chip which takes analog input and outputs a number of digital bits.

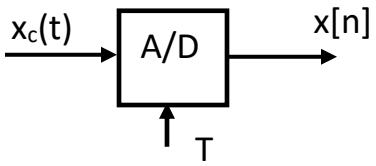


Fig2. Practical analog to digital converter.

The difference between the output signals in Fig1. And Fig2 is that the output in Fig1 is a discrete-time signal whose amplitude is continuous whereas the output in Fig2 is discrete both in time and the amplitude (hence digital signal).

Nyquist Theorem:

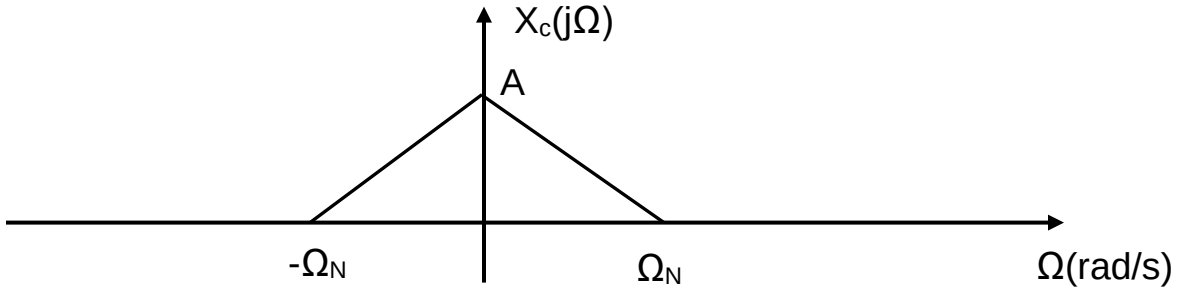
Let $x_c(t)$ be a bandlimited signal with $X_c(j\Omega) = 0$ for $|\Omega| \geq \Omega_N$. Then $x_c(t)$ is uniquely determined by its samples $x[n]=x_c(nT)$, $n=0,\pm1,\pm2,\ldots$ if $\Omega_s = \frac{2\pi}{T} \geq 2\Omega_N$.

Ω_s=sampling frequency

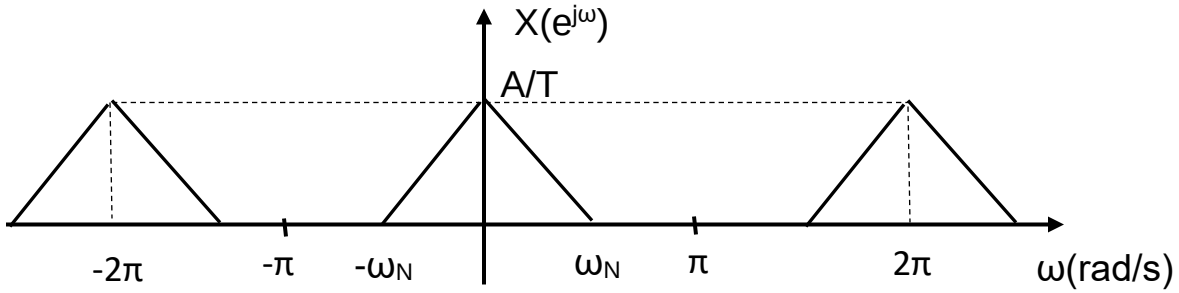
Ω_N=largest frequency of the analog signal

$2\Omega_N$=Nyquist Rate (minimum rate for sampling the signal)

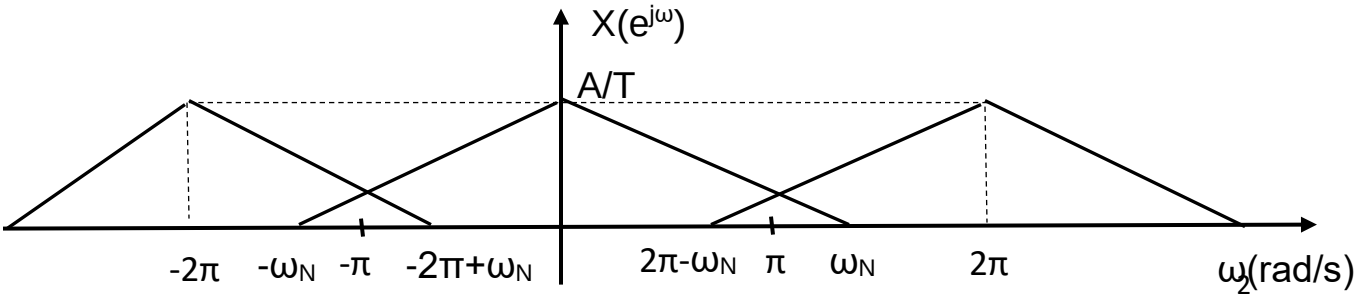
Consider an analog signal with CTFT spectrum given as below,



If there is no aliasing, then the DTFT spectrum becomes



If there is aliasing (loss of information), DTFT spectrum becomes



The DTFT of the signal x[n] is related with its CTFT as follows after samplling,

$$X(e^{j\omega}) = \frac{1}{T}\sum_{k=-\infty}^{\infty} X_c(j\frac{\omega - 2\pi k}{T})$$

CTFT:

$$X_c(j\Omega) = \int_{-\infty}^{\infty} x_c(t)e^{-j\Omega t}dt$$

$$x_c(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X_c(j\Omega)e^{j\Omega t}d\Omega$$

DTFT:

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

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

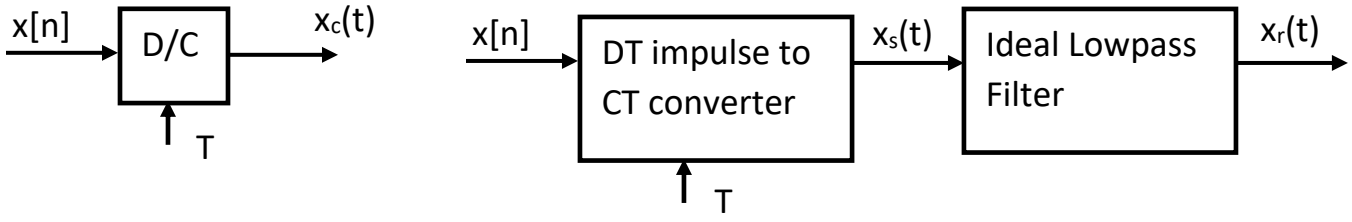
DFT:

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

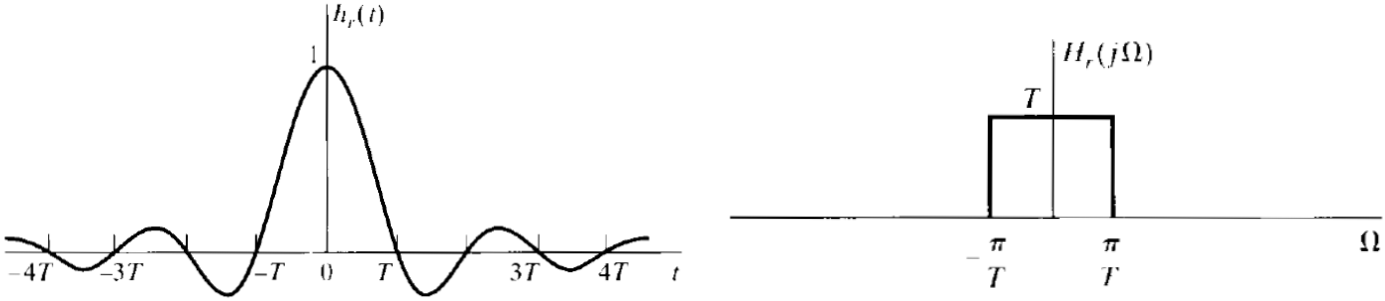
$$x[n] = \frac{1}{N}\sum_{k=0}^{N-1} X[k]e^{j\frac{2\pi}{N}kn}$$

Reconstuction of Bandlimited Signals

Digital signals should be converted to analog form to feed them through analog components such as loudspeakers, antennas, etc. Theoretically, signal reconstruction is done through the discrete to continuous converter block or D/C block. The practical counterpart of this is the digital to analog converter or D/A block.



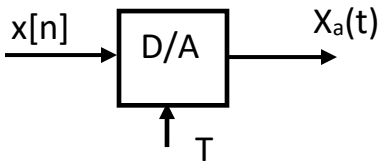
The ideal filter is described as a “brick-wall” filter with time and frequency characteristics given below



The reconstructed signal is $x_r(t)$ and given as,

$$x_r(t) = \sum_{n=-\infty}^{\infty} x[n] \frac{\sin(\frac{\pi(t-nT)}{T})}{\frac{\pi(t-nT)}{T}}$$

The practical signal reconstruction is performed with D/A converter, i.e.



In A/D and D/A operations, number of bits, maximum sampling frequency, SNR, SFDR, ENOB, THD are some of the important parameters. SNR for B bit A/D converter is approximately given as,

$$SNR = 6B + 1.76 \quad dB$$

Discrete Fourier Transform, DFT

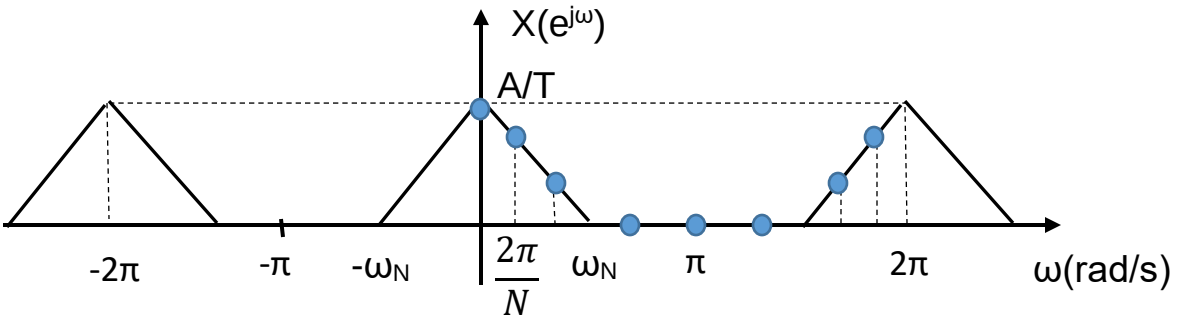
Discrete-time Fourier Transform, DTFT, is a continuous function of ω(rad). Hence it can not be realized in digital systems like a computer. Therefore there is a need to find a transform which is finite and discrete both in time and frequency. Discrete Fourier Transform, DFT, is obtained by sampling the DTFT and using the properties of DFS which is valid for periodic sequences. DFT can be seen as one period of DFS representation. The analysis and synthesis equations for DFT are given below.

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

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

DFT is defined for finite-length signals and the signal is assumed to be in [0, N-1].

$$X[k] = X\left(e^{j\omega}\right)\Big|_{\omega=\frac{2\pi}{N}k}, \quad k = 0,1,\dots,N-1$$



Ex: An analog bandlimited signal is sampled with fs=8kHz. N=256 samples of the signal are used to obtain the N-point DFT X[k], k=0,1,…,N-1. Determine the analog frequency corresponding to the k=16 point of the DFT.

Fast Fourier Transform, FFT

DFT is very useful to obtain the Fourier Transform of discrete-time signals. However it is computational expensive and efficiency is critical for practical applications. FFT is an algorithm to implement DFT in an efficient manner. The complexity of DFT is N^2 (complex multiplications) whereas the complexity of FFT is $\frac{N}{2}log_2 N$.

Ex: N=1024, N^2=1048576, $\frac{N}{2}log_2 N$=5120

Ex:
$$\{x[n]\}_{n=0}^{3} = [1 \quad 2 \quad 3 \quad 4]$$
$$\{X[k]\}_{k=0}^{3} = [10 \quad -2+j2 \quad -2 \quad -2-j2]$$

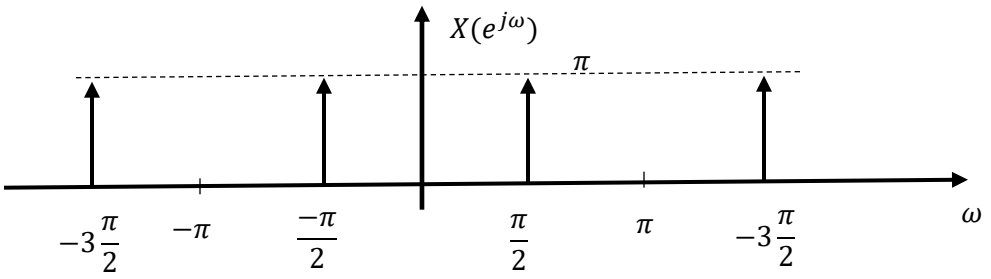
If fs=8kHz, find the analog frequency corresponding to each frequency bin of X[k].

Ex: Let $x_c(t) = \cos(2\pi 50t)$, and fs=200Hz

$$x[n] = x_c(nT) = \cos\left(2\pi 50\frac{n}{200}\right) = \cos(\frac{\pi}{2}n)$$

$$X(e^{j\omega}) = \pi \sum_{p=-\infty}^{\infty} \delta\left(\omega - \frac{\pi}{2} + 2\pi p\right) + \delta\left(\omega + \frac{\pi}{2} + 2\pi p\right)$$

Ex: If N=4 is selected and DFT is found we have,



$$x[n] = [1 \quad 0 \quad -1 \quad 0]$$

$$X[k] = [0 \quad 2 \quad 0 \quad 2]$$

Note that $\pi\delta\left(\omega - \frac{\pi}{2}\right)|_{\omega=\frac{\pi}{2}k} = \pi\delta\left(\frac{\pi}{2}(k-1)\right) = \pi\frac{1}{\frac{\pi}{2}}\delta(k-1) = 2\delta(k-1)$

What happens when N=5?

Ex: Repeat the above example for $x_c(t) = \sin(2\pi 50t)$

Ex: Computational complexity of filtering in time and frequency.

Let x[n] and h[n] be two N-point sequences. y[n]=x[n]*h[n] is a 2N-point sequence. We need to use 2N-point FFT to avoid aliasing. Hence the total complexity for the implementation in Fourier domain (using the complex multiplications as the reference) is

$$3Nlog_2(2N) + 2N$$

In time domain, the complexity is N^2. Let $N=2^n$ and

$$3Nlog_2(2N) + 2N < N^2$$

$$n < \frac{2^n - 5}{3}$$

$$3n + 5 < 2^n$$

$$a < b$$

n	a	b
4	17	16
5	20	32

From the above analysis, if x[n] and h[n] are both N-point complex sequences, their convolution will be implemented more efficiently in Frequency with FFT algorithm if N≥32.

Signal-to-Noise Ratio, SNR

SNR is a measure that compares the level of a desired signal to the level of noise. It is the most common term used in engineering applications to identify the quality of signal under consideration. It is usually defined as the ratio of the signal power to the noise power.

$$SNR = \frac{P_s}{P_n} = \frac{\sigma_s^2}{\sigma_n^2}$$

Signal and noise power should be considered in the same bandwidth. When the signal and noise are zero-mean and SNR is expressed in decibels,

$$SNR_{dB} = 10log_{10}(SNR) = 10log_{10}(\frac{\sigma_s}{\sigma_n})^2 = 20log_{10}(\frac{\sigma_s}{\sigma_n})$$

Note that for a particular time instant SNR can be approximated as the ratio of signal and noise voltage values.

Chirp and Signal Properties

Chirp is a signal whose frequency increase or decrease in time. It is especially used in radar, sonar and ultrasound. Consider a general sinusoid,

$$x(t) = Acos(\omega_0 t + \emptyset) = Acos(\theta(t))$$

Instantaneous frequency is

$$\omega(t) = \frac{d}{dt}\theta(t)$$

If $\theta(t) = 2\pi\mu t^2 + 2\pi f_0 t + \emptyset$, then

$$\omega(t) = 4\pi\mu t + 2\pi f_0$$

This corresponds to a chirp whose frequency increases in time. In this case,

$$f_1(t) = 2\mu t + f_0$$

is the final frequency.

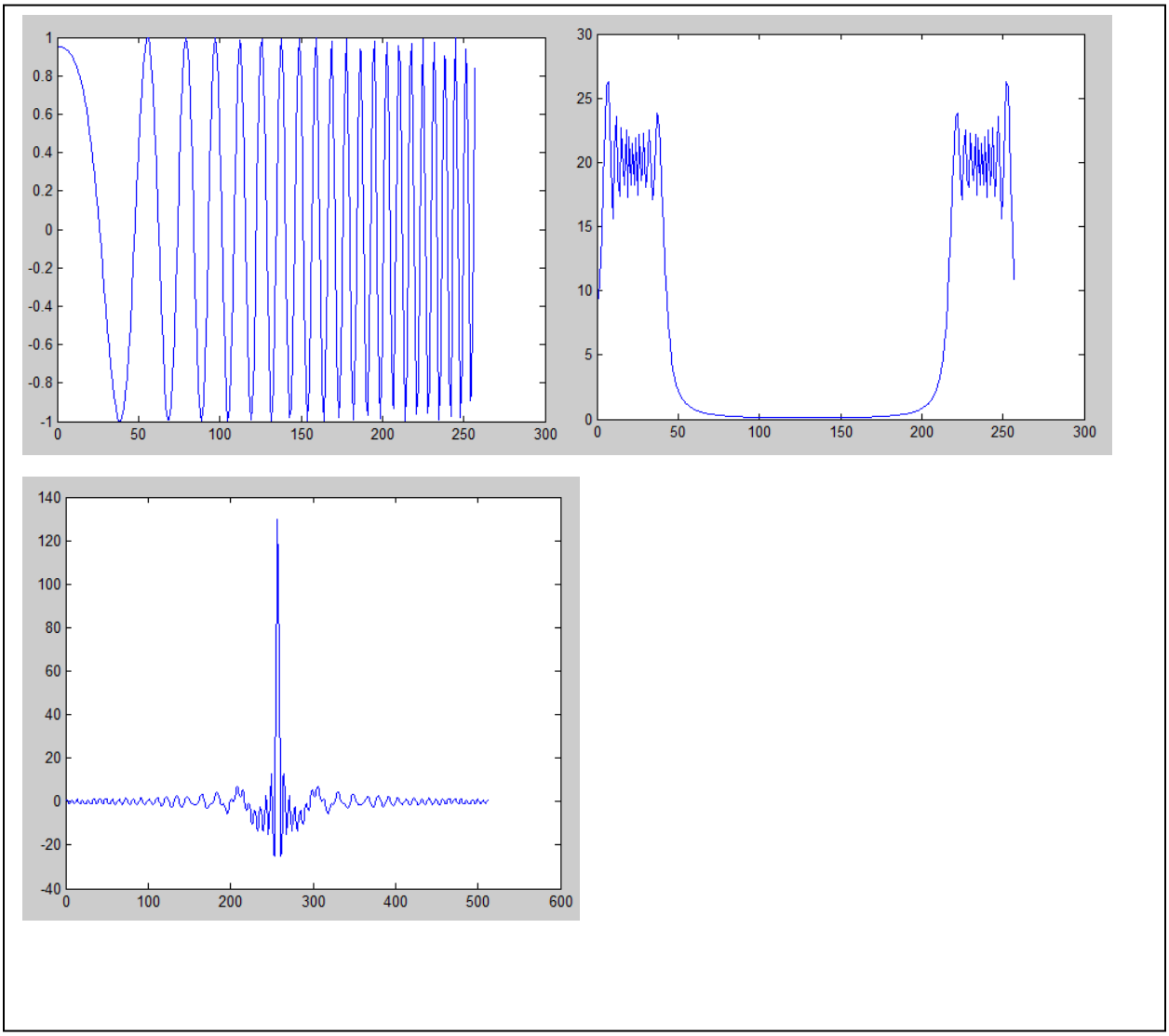
$$k = 2\mu = \frac{f_1 - f_0}{T}$$

k is the rate of frequency increase over duration T.

$$x(t) = \cos(2\pi \frac{k}{2} t^2 + 2\pi f_0 t)$$

x(t) is a chirp starting from f_0 to f_1 frequency.

An example of a chirp in time and frequency is given below. The auto-correlation function is also given below.



Correlation Property: When a linear frequency modulated (LFM) chirp is correlated with itself, a pulse is obtained. This is an energy compaction called as the pulse compression.

Ex: The Matlab script for generating the above chirp signal is given below.

```
t=[0:256];
w0=pi/10;
a=0.002*t.*t;
x=cos(a+w0);
y=abs(fft(x));
figure
plot(x)
figure
plot(y)

c=conv(x,fliplr(x))
plot(c)
```

Digital Filters

Digital filters are used to remove certain frequency bands from the original signal to clear the signal from noise and interference. While there are different types of digital filters, we will concentrate on simple, deterministic, fixed coefficient digital filters in this part. One counter example to this type of filters is the Wiener Filter whose coefficients are dependent on the input signal statistics.

There are two types of digital filters:

a) Finite-duration impulse response (FIR) filters
b) Infinite-duration impulse response (IIR) filters

FIR filter is a filter whose impulse response is finite duration, or h[n] is nonzero only in [0,N-1]. FIR filters are always stable and linear-phase characteristics can easily be obtained making them one of the best choice in image processing.

IIR filter is a filter whose impulse response is infinite duration. They can be unstable so filter coefficients should be designed carefully. The main advantage of the IIR filters is that it is possible to design highly selective frequency response with a couple of coefficients. In other words, they are computationally effective for designing sharp filters. IIR causal and stable filter have always nonlinear phase. In audio processing, phase characteristics is usually not very critical and hence IIR filters are frequently used in such applications.

Filter input-output relation is given as a difference equation in time.

FIR:

$$y[n] = \sum_{k=0}^{K-1} h[k]x[n-k]$$

$$h[n] = \sum_{k=0}^{K-1} h[k]\delta[n-k]$$

IIR:

$$\sum_{m=0}^{M-1} a_m y[n-m] = \sum_{k=0}^{K-1} b_k x[n-k]$$

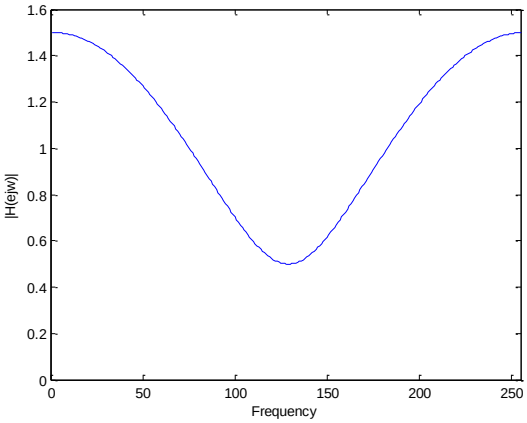
$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{k=0}^{K-1} b_k z^{-k}}{\sum_{m=0}^{M-1} a_m z^{-m}}$$

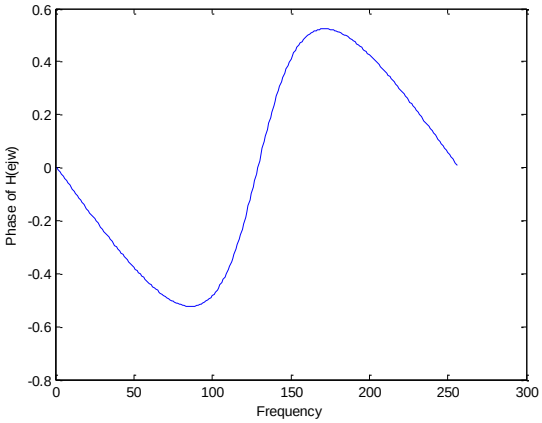
Ex: (Nonlinear Phase FIR Filter)

FIR filter is given as,

$$h[n] = \delta[n] + \frac{1}{2}\delta[n-1]$$

$$H\left(e^{j\omega}\right) = 1 + \frac{1}{2}e^{-j\omega} = \frac{1}{2} + e^{-j\frac{\omega}{2}}\cos\left(\frac{\omega}{2}\right)$$

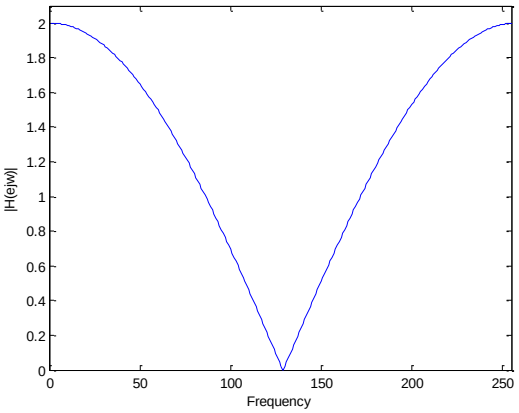


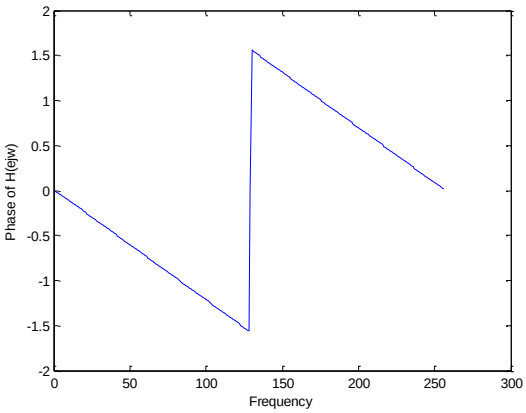


Ex: (Linear-phase FIR Filter)

$$h[n] = \delta[n] + \delta[n-1]$$

$$H\left(e^{j\omega}\right) = 1 + e^{-j\omega} = 2e^{-j\frac{\omega}{2}}\cos\left(\frac{\omega}{2}\right)$$



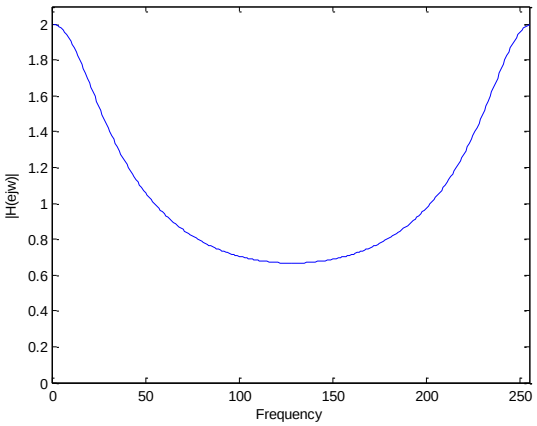


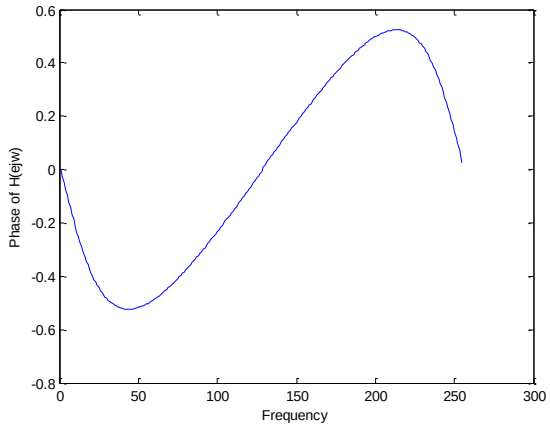
Ex: IIR filter is given as

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

$$h[n] = \left(\frac{1}{2}\right)^n u[n]$$

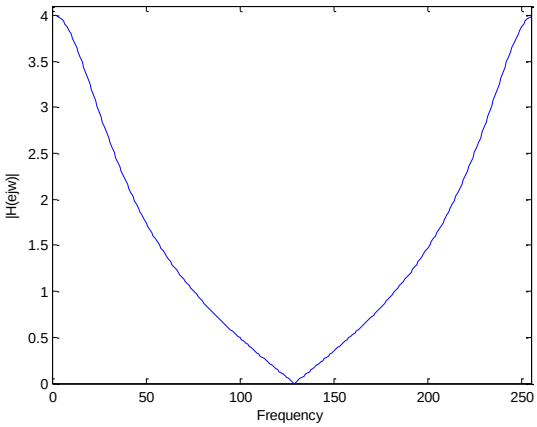
$$H(e^{j\omega}) = \frac{1}{1 - \frac{1}{2}e^{-j\omega}}$$

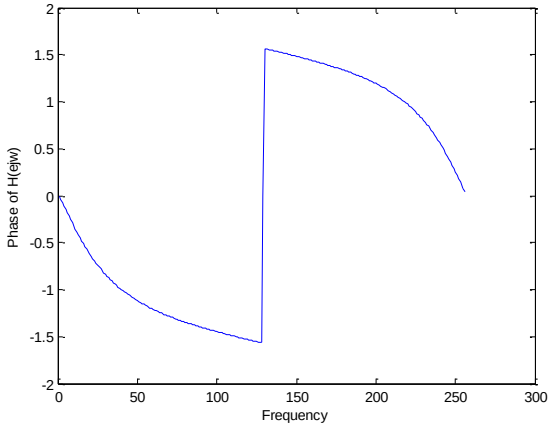




Ex: IIR filter is given as

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

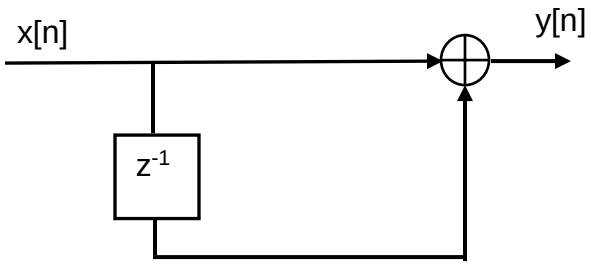




FIR Implementations

A.Direct Implementation: Direct Form I

Ex: $h[n] = \delta[n] + \frac{1}{2}\delta[n-1]$



Above implementation requires N^2 multiplications in time for N-point sequences of x[n] and h[n].

B. Fourier Domain Implementation

Y[k]=H[k]X[k] and y[n] is the 2N-point IDFT. MATLAB implementation for this is given below

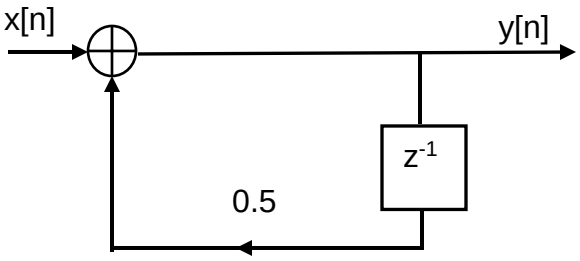
```
X=fft(x,2N);

H=fft(h,2N);

Y=X.*H;

Y=ifft(Y,2N);
```

This requires $3Nlog_2(2N) + 2N$ complex multiplications. Fourier domain filter implemenation is more efficient as the length of the sequences increases.

IIR Implementations

IIR filters are especially preferred due to their efficient implementations and sharp filter characteristics. The output can be computed recursively.

Ex: $y[n] = \frac{1}{2}y[n-1] + x[n]$



Above filter requires only one multiplication per sample and the filter has infinite length!

FIR Filter	Advantages	Disadvantages
	Always stable	Computationally expensive
	Can have linear phase	

IIR Filter	Advantages	Disadvantages
	Computationally Efficient	May be unstable
	Sharp filters can be designed	Has nonlinear phase

IIR Filter Implementation in FPGA

In FPGA, we can only implement fixed-point operations. One convenient way is the use of integer arithmetic in FPGA. Then we are faced with the problem of converting real numbers to integer representation. Assume that $x_1[n]\ \epsilon\ [-1,1]$.

B: number of bits in the representation.

Assuming that 1 bit is used in sign representation,

$$x_2[n] = floor(x_1[n]2^{B-1})$$

$$x_2[n]\ \epsilon\ [-2^{B-1}, 2^{B-1}]$$

For unsigned representation,

$$x_2[n] = floor((1 + x_1[n])2^B)$$

Consider a first order IIR filter

$$y[n] = \frac{1}{2}y[n-1] + x_1[n]$$

For FPGA input and the coefficient should be integer and hence we have

$$y[n] = \left\lfloor \frac{1}{2}2^{(B-1)} \right\rfloor y[n-1] + x_2[n]$$

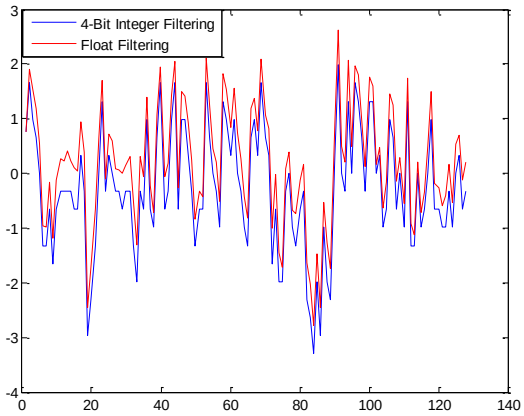
Here $\lfloor\ \ \rfloor$ is the floor operator.

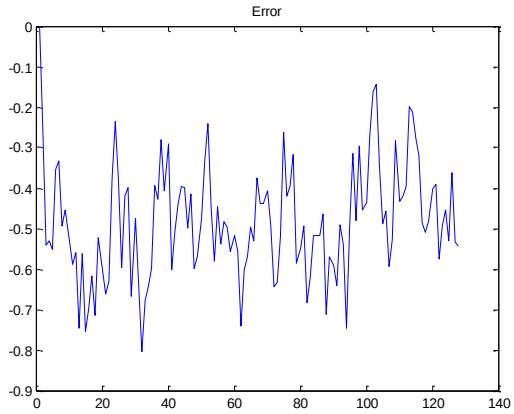
Note that two numbers in B bits are multiplied resulting a number with 2B bits and it should be rounded back to B bits.

$$y[n] = \left\lfloor \left\lfloor \frac{1}{2}2^{(B-1)} \right\rfloor y[n-1]2^{-(B-1)} \right\rfloor + x_2[n]$$

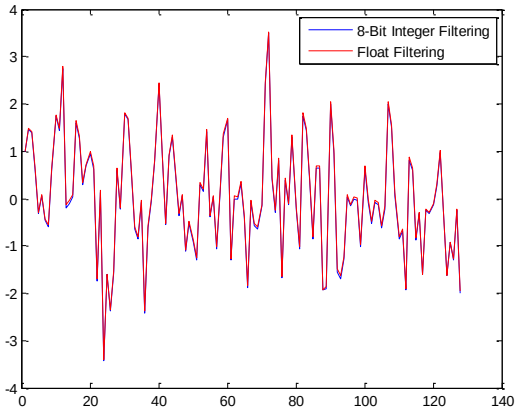
The results for integer and float filtering are given below for the above filter,

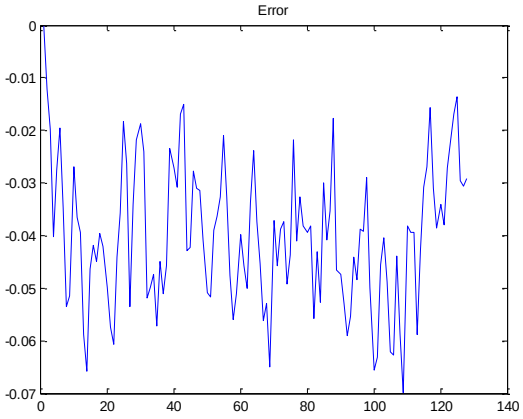
4-Bit integer filtering





8-Bit integer filtering



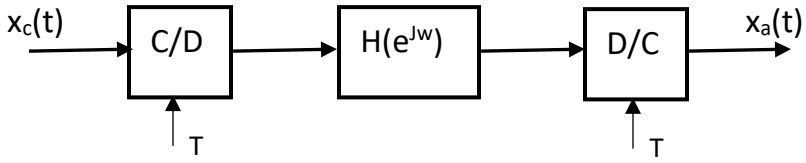


Below a Verilog IIR Filter code is provided for the above 1st order filter. Note that input is 16bits and the output is 32bits. Determine and write the filter difference equation for the following code.

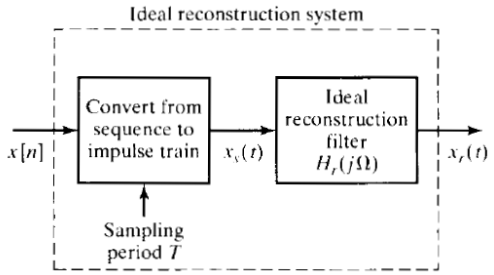
```
module iir(input signed[15:0] xn, input clk, rst_n, ouput reg signed [31:0] yn);
        reg signed [31:0] yn-1;
        always@ (yn-1 or xn) yn=(yn-1 >>>1)+xn;
       always@(posedge clk or negedge rst_n) begin
            if (!rst_n)
                    yn-1<=0;
            else
                   yn-1 <=yn;
             end
        endmodule
```

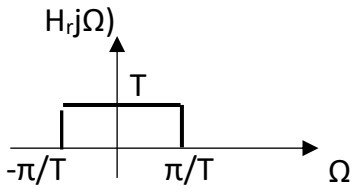
Digital Signal Processing Structures

Most of the digital systems have a generic system structure as shown below. First analog signal is converted to discrete-time form. This is achieved in C/D block. Then, discrete-time signal is processed by the digital system where in the following figure there is a simple LTI filter for this purpose. The output of the filter is converted to analog by using D/C block. Usually we do not have sampling rate change at the input and output and hence sampling period, T, is same for both C/D and D/C converters.

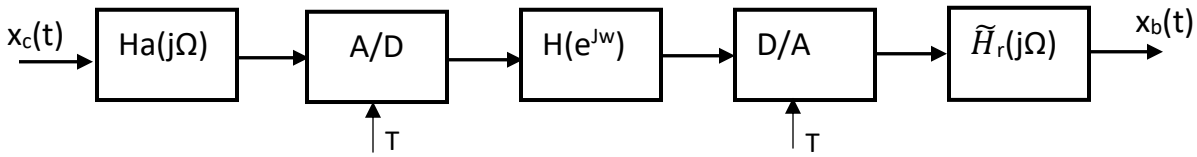


D/C block has the following equivalent structure which includes an ideal reconstruction filter, $H_r(j\Omega)$.





A practical counterpart of a digital system structure is given below. In this case, there is an analog anti-aliasing filter to limit the highest frequency of the analog signal so that there is no aliasing after sampling. Note that aliasing correspond to loss of information and cannot be corrected once it occurred. Therefore it is usually better to limit the highest frequency of the analog signal before it is being sampled. C/D and D/C blocks are replaced by A/D and D/A blocks respectively. Analog-to-digital converter is usually a chip which has an analog input (Ex. Between [0,5V] or [-5V, 5V]) and digital output. Digital output may be parallel or series depending on the type of the A/D converter. The number of bits for the A/D converter determines the quality of the conversion and SNR of the digital signal. Digital-to-analog converter, D/A is also a chip with digital input and analog output. Note that there is no aliasing problem in D/A operation. The number of bits in D/A converter also determines the SNR at the output signal. Usually D/A chips are faster than A/D chips. Currently, we have 8-bit, 1 GSPS and better A/D converters. Since D/A converter uses pulses rather than impulses for the reconstruction, compensated reconstruction filter, $\widetilde{H}_r(j\Omega)$, may be used to correct the frequency response.



We should note that the signal average power does not change after sampling, i.e.,

$$E\{x_c^2(t)\} = E\{x^2[n]\}, \quad \forall t, n$$

Signal energy is an important parameter and we can find it in different ways. Parseval's Relation is a tool to find signal energy both in time and frequency. For DFT case,

$$\sum_{n=-\infty}^{\infty} |x[n]|^2 = \frac{1}{N}\sum_{k=0}^{N-1} |X[k]|^2$$

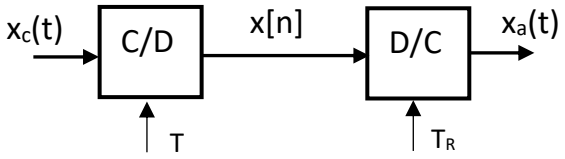
For DTFT case,

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

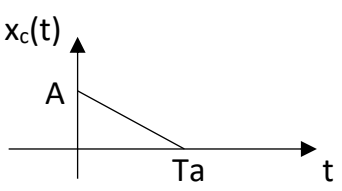
Above may be explained as summing power-per-sample across time is equal to the integration of spectral power over frequency. Another version of Parseval's relation is given below.

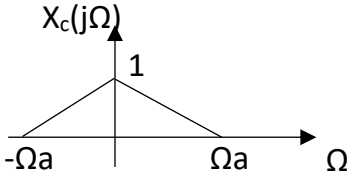
$$\sum_{n=-\infty}^{\infty} x[n]y^*[n] = \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{jw})Y^*(e^{jw})\, d\omega$$

The use of Parseval's relation can be shown in an example where we have cascade connected C/D and D/C blocks with different sampling periods.



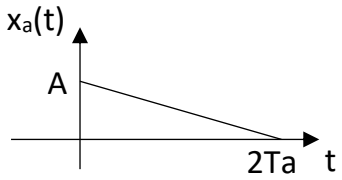
Let $x_c(t)$ be a bandlimited analog signal which is sampled by a sampling rate above the Nyquist rate.

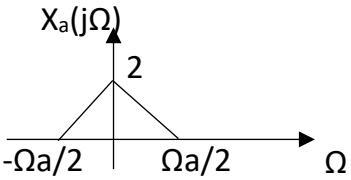




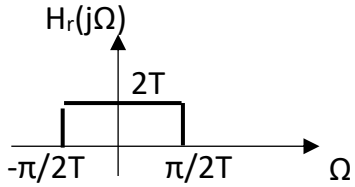
If T_R=T, $x_a(t)$=$x_c(t)$ and both have the same signal energy E_c.

If T_R=2T, $x_a(t)=x_c(t/2)$ and $X_a(j\Omega) = 2X_c(j2\Omega)$





Note that for T_R=2T



Then, signal energy becomes

$$E_a = \int_{-\infty}^{\infty} x_a^2(t)dt = \int_{-\infty}^{\infty} x_c^2(t/2)dt = \int_{-\infty}^{\infty} x_c^2(u)2du = 2\int_{-\infty}^{\infty} x_c^2(u)du = 2E_c$$

which is twice the previous signal energy.

Ex:You can try the following MATLAB code for discrete-time version of this problem

```
x=[sin(pi/10*[0:19])];
e=sum(x.^2);
y=interp(x,2);
e2=sum(y.^2)
```

Uncertainty Principle: (Heisenberg's uncertainty principle)

For any function with time-duration, Δt , and frequency bandwidth, Δf , following limit is valid, i.e.,

$$\Delta t \Delta f \geq 1$$

This leads to following result. If a function is limited in time, it is not limited in frequency and vice versa. The optimum function which is most compact both in time and frequency is a Gaussian function.

Discrete-time Random Signals and Correlation

Until now, we have assumed that the signals are deterministic, i.e, that each value of a sequence is uniquely determined by a mathematical expression, a table of data or a rule of some type. In many situations, the processes that generate signals are complex as to make a precise description of a signal extremely difficult if not impossible. In such cases, modeling a signal as a stochastic process is analytically useful. For example, speech signal for automatic speech recognition is usually modeled as a stochastic process to extract the information content.

Random Variable: A random variable is a function that maps the outcomes of an experiment to a real number.

Random variable is used to model real events with their mathematical representations.

Random Process: A random process is a collection of random variables where at each time instant we have a random variable.

Random process can be used to represent signals in time to model them with their statistical characteristics.

Wide-Sense Stationarity (WSS): A random process is WSS if its mean is constant and autocorrelation function depends only on the time difference, i.e.

$$m_x = E\{x[n]\} = c$$

$$R_x[m_1, m_2] = R_x[m_2 - m_1]$$

Auto-correlation function can be written as,

$$R_x[m] = E\{x[n]x[n-m]^*\}$$

Similarly, cross-correlation and cross-covariance functions can be written as,

$$R_{xy}[m] = E\{x[n]y[n-m]^*\} = C_{xy}[m] + m_x m_y^*$$

$$C_{xy}[m] = E\{(x[n] - m_x)(y[n-m] - m_y)^*\}$$

Consider sample (or deterministic) correlation function

$$\hat{R}_x[l] = \frac{1}{N_s}\sum_{n=0}^{N_s-l-1} x[n+l]x^*[n] = \frac{1}{N_s}\sum_{n=l}^{N_s-1} x[n]x^*[n-l] \qquad 0 \le l < N_s$$

Similarly, sample cross-correlation function can be written as,

$$\hat{R}_{xy}[l] = \frac{1}{N_s}\sum_{n=0}^{N_s-l-1} x[n+l]y^*[n] = \frac{1}{N_s}\sum_{n=l}^{N_s-1} x[n]y^*[n-l]$$

Deterministic Correlation and Its Relation to Convolution

In the previous part, we have defined the theoretical and sample definitions for cross-correlation. More specifically the sample cross-correlation is written as,

$$\hat{R}_{xy}[l] = \frac{1}{N_s}\sum_{n=0}^{N_s-l-1} x[n+l]y^*[n] = \frac{1}{N_s}\sum_{n=l}^{N_s-1} x[n]y^*[n-l]$$

We would like to relate this expression with convolution. Consider the convolution of two N-point sequences, ($0 \leq n < N$)

$$x[n] * y^*[-n] = \sum_{m=0}^{N-1} x[m]y^*[-(n-m)] = \sum_{m=n}^{N-1} x[m]y^*[m-n]$$

Note that y[n] is also a sequence which is nonzero in 0≤n<N-1.

Therefore cross-correlation can be written in terms of convolution as,

$$\hat{R}_{xy}[l] = \frac{1}{N_s} x[n] * y^*[-n]$$

The DTFT of the above expression is the power spectral density function, and it is given as,

$$\hat{S}_{xy}(e^{j\omega}) = \frac{1}{N_s} X(e^{j\omega}) Y^*(e^{j\omega})$$

Note that if we consider the deterministic auto-correlation, we have,

$$\hat{R}_{xx}[l] = \frac{1}{N_s} x[n] * x^*[-n]$$

The power spectral density function of x[n] becomes,

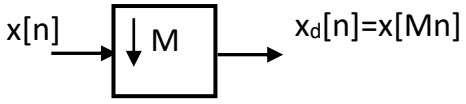
$$\hat{S}_{xx}(e^{j\omega}) = \frac{1}{N_s} X(e^{j\omega}) X^*(e^{j\omega}) = \frac{1}{N_s} \left|X(e^{j\omega})\right|^2$$

Sampling Rate Change: Decimation and Interpolation

Sampling rate change in digital systems is performed through decimation and interpolation.

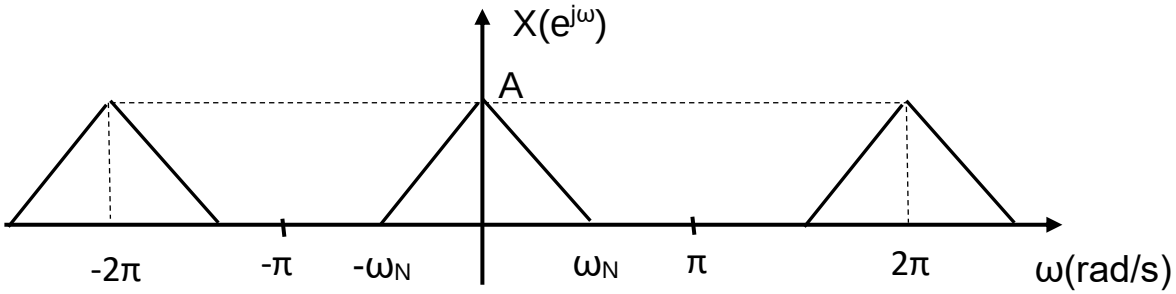
A.Decimation (Downsampling)

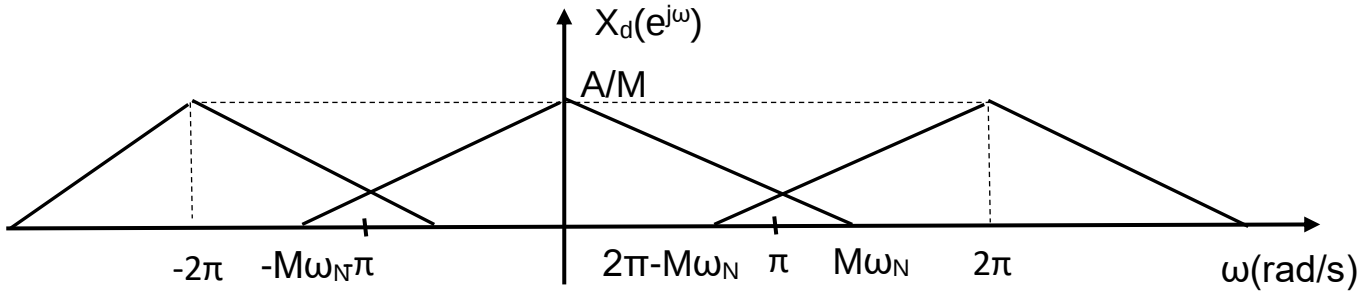
In decimation, signal samples are reduced by an integer amount. The following block corresponds to a compressor where only one out of M samples are passed to the output.



$$X_d(e^{jw}) = \frac{1}{M}\sum_{i=0}^{M-1} X\left(e^{j(\frac{w-2\pi i}{M})}\right)$$

It is possible to have aliasing after decimation operation. $M\omega_N<\pi$ is the condition for no aliasing.





Ex: x[n]=[1 2 3 4 5 6 7 8 9]

M=2, $x_d[n]=[1\ \ 3\ \ 5\ \ 7\ \ 9]$

Downsampling Properties:

a)Linear

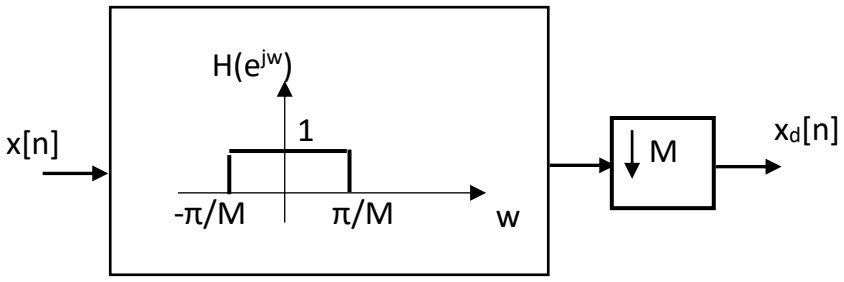
b)Time-varying

c)Signal average power does not change

d)$x_d[n]$ is WSS if x[n] is WSS.

e)Energy is reduced.

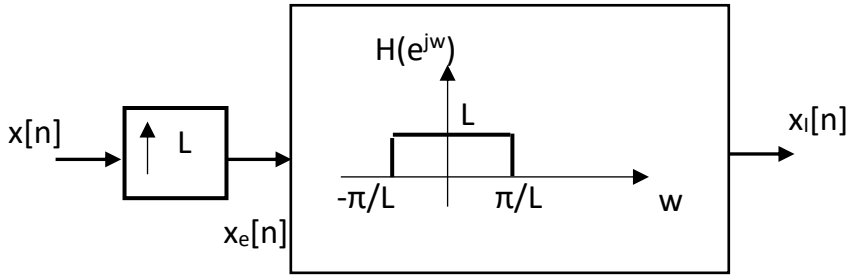
Structure For Decimation



In decimation, we first filter the input signal to make sure that it has limited bandwidth so that there is no aliasing after decimation. Note that the signal bandwidth enlarges by a factor of M in the compressor block.

B.Interpolation (Upsampling)

In interpolation, signal samples are increased by an integer amount. The expander block below insert L-1 zeros in between every two samples of x[n] and $x_e[n]$ is obtained. After filtering by h[n], zeros are filled with the interpolated values and the resulting signal $x_I[n]$ is obtained. The amplitude of the filter has a value L. This is to match with the new sampling rate T/L at the filter output.



$$x_e[n] = \begin{cases} x\left[\frac{n}{L}\right], & n = 0, \pm 1, \pm 2, \ldots \\ 0, & otherwise \end{cases}$$

$$X_e(e^{jw}) = X(e^{jwL})$$

Ex: x[n]=[1 2 3 4 5]

L=2, $x_e[n]$=[1 0 2 0 3 0 4 0 5]

Upsampling Properties:

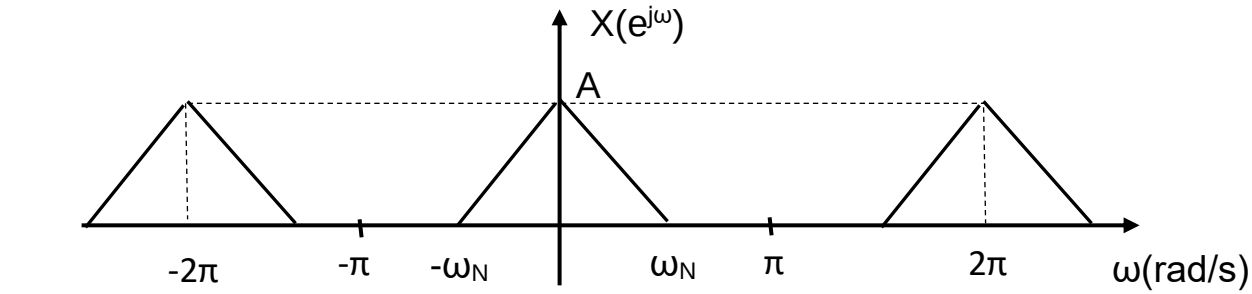
a)Linear

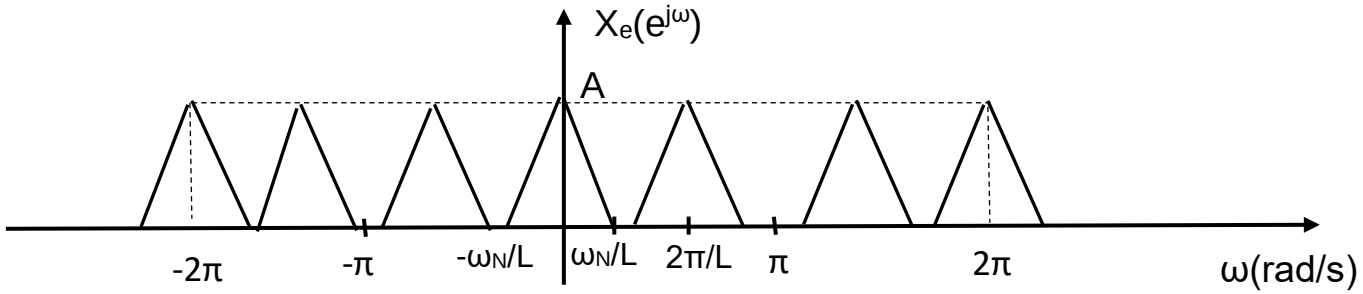
b) Time-varying

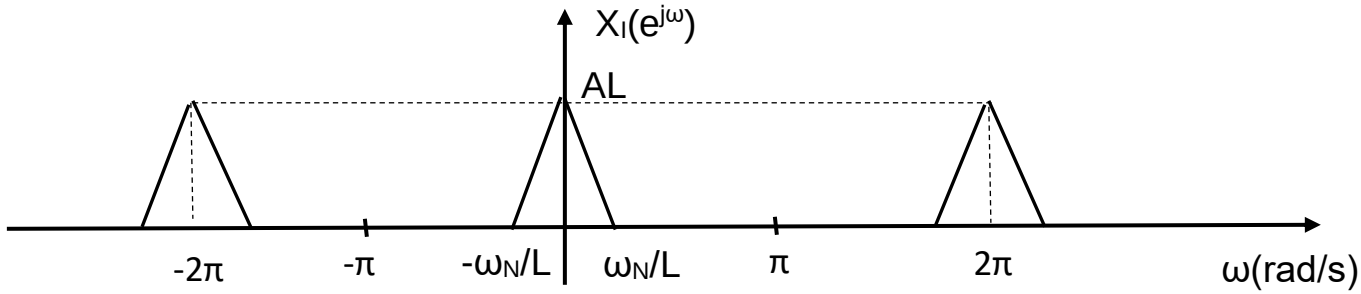
c) xI[n] is <u>not</u> WSS if x[n] is WSS.

d) Signal average power changes

e) Energy increases.

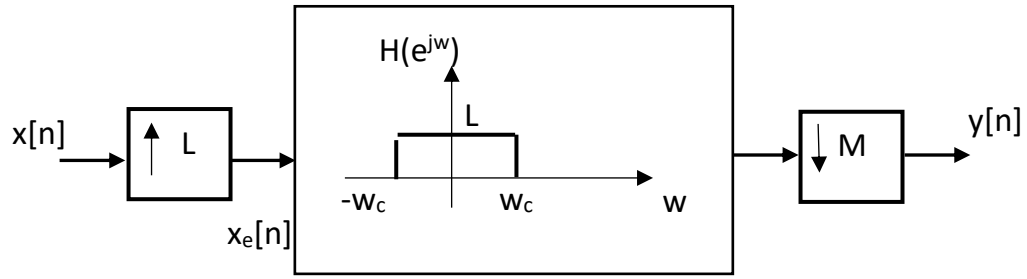






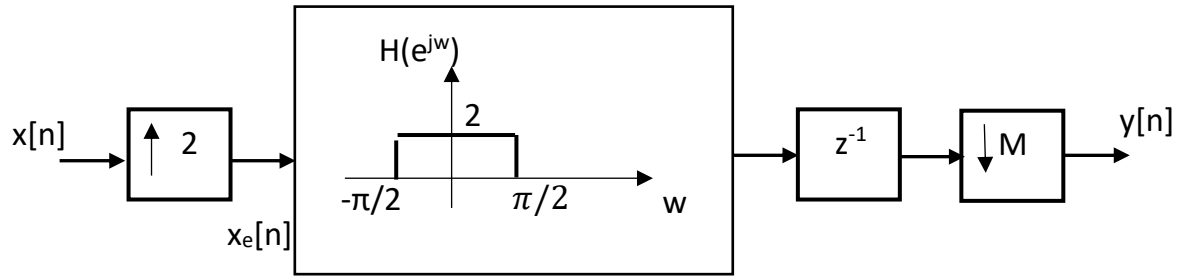
C. Sampling Rate Change By Non-integer Factor

In communications systems, different signals with different sampling rates are combined as a single channel to transmit from the central hub. This is done by changing the sampling rates of each signal to a common rate. Usually a simple integer change is not sufficient and a non-integer sampling rate is applied. Non-integer sampling rate is not arbitrary and it should be a rational number for implementation.

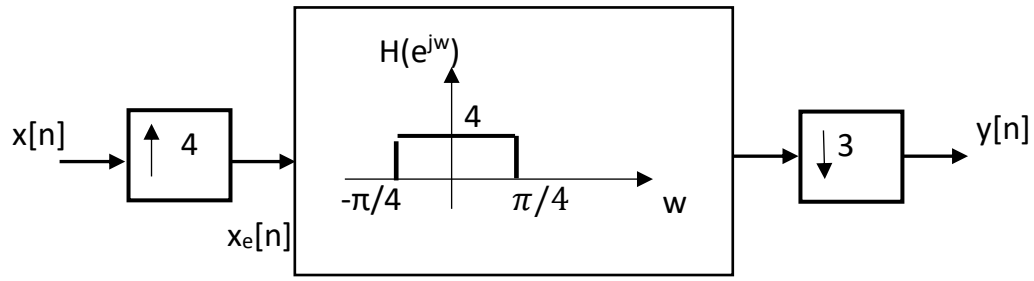


$$\omega_c = min\left\{\frac{\pi}{L}, \frac{\pi}{M}\right\}$$

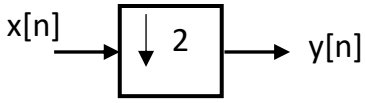
Ex1: Design a system which delays its input by 1/2 sample. (No rate change)



Ex2: Design a system which increases the sampling rate by 4/3 times. (i.e., 4/3fs, L=4, M=3)



Compressor and expander blocks are linear but time varying. Consider



x[n]=[1 2 3 4 5] y[n]=[1 3 5]

x[n-1]=[0 1 2 3 4 5] y[n]=[0 2 4]

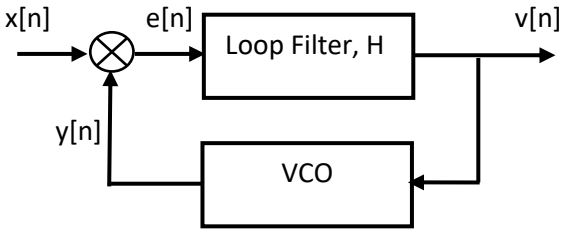
Digital Phase-locked Loop, PLL

PLLs are used in signal processing, communications, multimedia, etc. PLL locks onto the incoming signal frequency and phase if the internally generated sinusoid frequency is sufficiently close to the incoming signal. This means that if the input signal frequency is within the lock-in range, PLL locks to the input frequency. While PLL fundamentally locks to the signal phase, it can also compensate small frequency deviations. Therefore the function of PLL is to replicate and track the frequency and phase of the input.

PLL structure is nonlinear in general. Usually linear approximations are used for mathematical analysis. PLL consists of three major components: a multiplier, a loop filter and a voltage-controlled oscillator (VCO) connected in the form of a feedback system. VCO is a sinusoidal generator whose frequency is determined by a voltage applied to it from an external source. In effect, any frequency modulator may serve as a VCO. We assume that initially we have adjusted the VCO so that when the voltage is zero, two conditions are satisfied:

1)The frequency of the VCO is precisely set at the unmodulated carrier frequency f_c.

2)The VCO output has a 90 degree phase-shift with respect to the unmodulated carrier wave.



Let $$x[n] = A_1 \sin(2\pi f_1 n + \theta_1)$$

$$y[n] = A_2 \cos(2\pi f_0 n + \theta_2)$$

Then

$$e[n] = \frac{1}{2}\left[A_1 A_2 \sin\left(2\pi(f_1 - f_0)n + (\theta_1 - \theta_2)\right) + A_1 A_2 \sin(2\pi(f_1 + f_0)n + (\theta_1 + \theta_2)\right]$$

Loop filter $H(e^{jw})$ is a lowpass filter with gain k_m.

$$y[n] = \frac{1}{2}\left[A_1 A_2 k_m \sin\left(2\pi(f_1 - f_0)n + (\theta_1 - \theta_2)\right)\right]$$

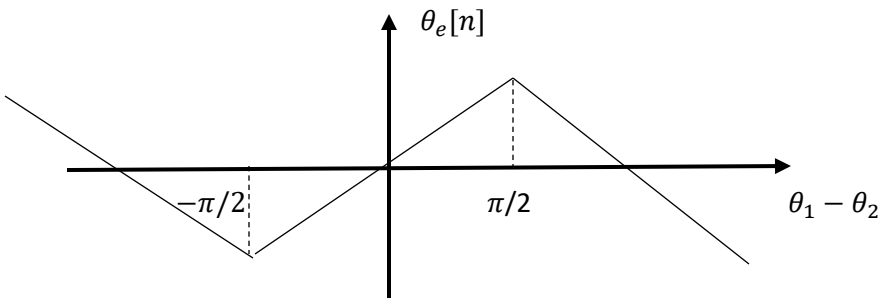
Define loop gain as, K_0

$$K_0 = \frac{1}{2} A_1 A_2 k_m$$

If we define

$$\theta_e[n] = 2\pi(f_1 - f_0)n + \theta_1 - \theta_2$$

Consider the case where $f_1 = f_0$, then $\theta_e[n]$ has a relation with the phase difference given as below.



If $|f_1 - f_0| < \frac{1}{2}K_0$ then frequency lock is possible and

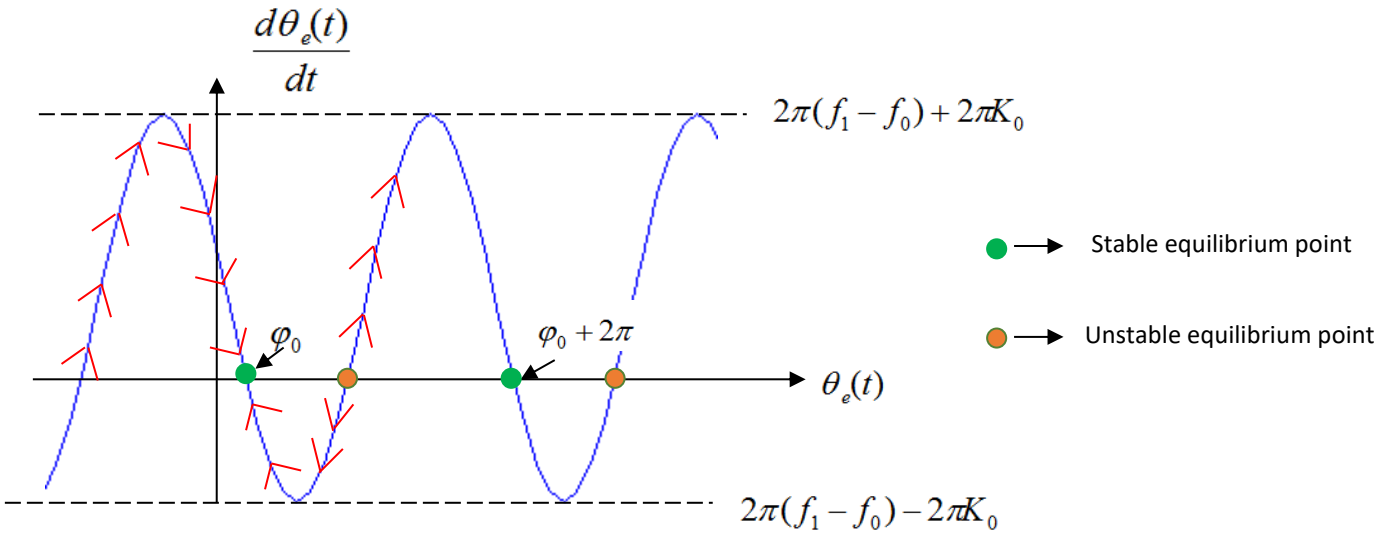
$[f_0 - \frac{1}{2}K_0,\ \ f_0 + \frac{1}{2}K_0]$ is the lock-in range of PLL.

Note that loop gain K_0 can not be selected very large. If K_0 is large, PLL can be unstable and can not achieve lock. K_0 should not be selected very small either. If it is too small, then lock-in range of PLL becomes small and small deviations in frequency or phase may lead to out of lock situations. Hence an appropriate value for K_0 should be selected based on the intended frequency bandwidth of operation. While there are certain mathematical limits for PLL gain selection, they do not have simple forms and are outside the scope of this course. Therefore a trial and error approach for the gain selection can be used in the laboratory experiments.

If we consider continuous time treatment of the PLL, error term derivative can be expressed as,

$$\frac{d\theta_e(t)}{dt} = 2\pi(f_1 - f_0) - 2\pi K_0 \sin(\theta_e(t)).$$

Using the above equation, we can plot the phase trajectory as shown in the following figure,



Note that green points in the above figure correspond to stable equilibrium points whereas orange points are unstable equilibrium points. If the derivative of phase error is positive, it attempts to increase as shown by red arrows. Otherwise, it decreases.

When the error is small, we can have an approximate expression for phase error as follows, i.e.,

$$\sin(\theta_e[n]) \approx \theta_e[n]$$

This can be used to linearize the nonlinear PLL equations. Note that linearization of PLL equations are useful to better understand the PLL operation. However, nonlinear systems are in general not easy to characterize completely. While linear approximations may be used for implementing nonlinear systems, there is no guarantee that these will perform better than the original nonlinear system.

The response for a digital PLL is given in Figure 1. In this figure, PLL compensates for a small frequency and phase difference in time and locks to the incoming signal. As it is seen in Figure 1, there is large error at the beginning which is decreased in time showing a stable PLL operation. In Figure 2, the error decreases in time approaching to a very small value after convergence is achieved.

Acquisition Mode: When a phase-locked loop is used for coherent detection, the loop must first lock onto the input signal and then follow the variations of its phase angle with time. The process of bringing a loop into phase-lock is called acquisition and the following process to keep the variations small is called tracking. During the acquisition, angle variations might be large mandating the use of nonlinear model for the PLL. However nonlinear analysis of the PLL loop is beyond the scope of this course.

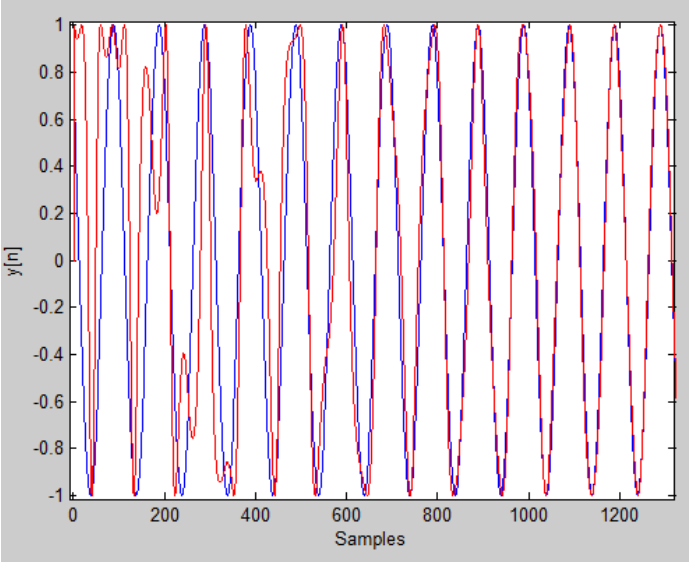


Figure 1. PLL lock-in process in time. Blue is the target signal for lock-in, red is the output of digital PLL.

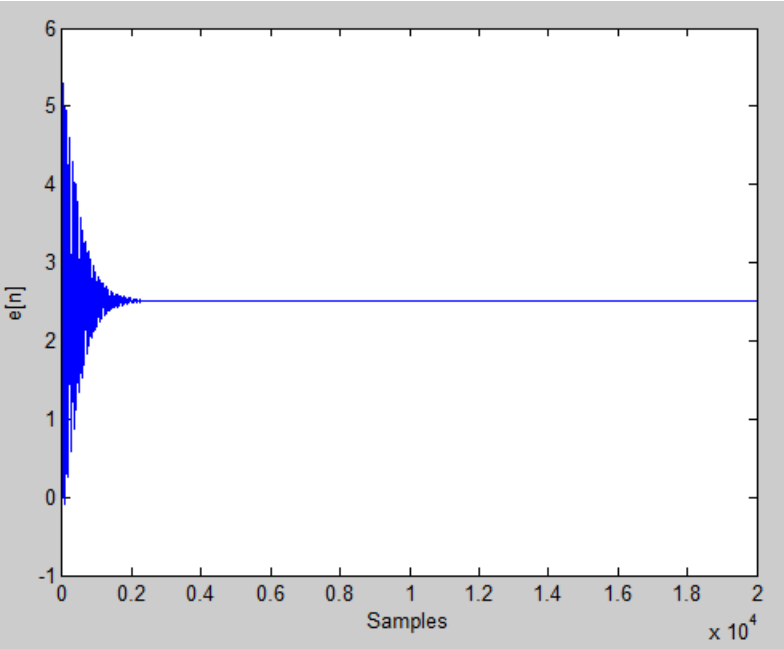


Figure 2: PLL error in time.

INTRODUCTION TO BASIC CONCEPTS IN STATISTICAL SIGNAL PROCESSING

In this part, a basic background in statistical signal processing is presented. This background is used fort he following Chapters such as Optimum Filtering (Wiener Filtering) and Adaptive Signal Processing.

Random Variable: A random variable is a function that maps the outcomes of an experiment to a real number.

Random variable is used to model real events with their mathematical representations.

Random Process: A random process is a collection of random variables where at each time instant we have a random variable.

Random process can be used to represent signals in time to model them with their statistical characteristics.

Random Vector: It is a vector where the elements are random variables. Consider the following random vector,

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ \dots \\ x_N \end{bmatrix}$$

Mean of a random vector, $\mathbf{m}_x$, can be written as,

$$\boldsymbol{m}_x = E\{\boldsymbol{x}\} = \int_{-\infty}^{\infty} \boldsymbol{x} f_x(\boldsymbol{x}) d\boldsymbol{x} = \begin{bmatrix} m_1 \\ \dots \\ m_N \end{bmatrix}$$

Correlation matrix is, $\mathbf{R}_x$,

$$\boldsymbol{R}_x = E\{\boldsymbol{x}\boldsymbol{x}^H\} = \begin{bmatrix} E\{|x_1|^2\} & \dots & E\{x_1 x_N^*\} \\ \dots & \dots & \dots \\ E\{x_N x_1^*\} & \dots & E\{|x_N|^2\} \end{bmatrix}$$

Covariance matrix is, $\mathbf{C}_x$,

$$\boldsymbol{C}_x = E\{(\boldsymbol{x} - \boldsymbol{m}_x)(\boldsymbol{x} - \boldsymbol{m}_x)^H\} = \boldsymbol{R}_x - \boldsymbol{m}_x \boldsymbol{m}_x^H$$

Cross Covariance, $\mathbf{C}_{xy}$,

$$\boldsymbol{C}_{xy} = E\{(\boldsymbol{x} - \boldsymbol{m}_x)(\boldsymbol{x} - \boldsymbol{m}_x)^H\} = \boldsymbol{R}_{xy} - \boldsymbol{m}_x \boldsymbol{m}_y$$

Two random vectors are uncorrelated if

$$\boldsymbol{R}_{xy} = E\{\boldsymbol{x}\boldsymbol{y}^H\} = E\{\boldsymbol{x}\} E\{\boldsymbol{y}^H\} = \boldsymbol{m}_x \boldsymbol{m}_y^H$$

Hence $\boldsymbol{C}_{xy} = 0$ for uncorrelated random vectors.

Two random vectors are orthogonal if

$$\boldsymbol{R}_{xy} = E\{\boldsymbol{x}\boldsymbol{y}^H\} = 0$$

In statistical signal processing, second order statistics is the most powerful tool since it can be estimated with small number of samples and at low SNR. Second order statistics involves mean and covariance of a signal. Hence second order statistical inference use the mean and covariance terms of a signal. Usually if the mean of the signal is not zero, it is subtracted from the signal and a zero-mean signal is obtained. Signal correlation and covariance are the same for a zero-mean signal.

In the above part, we have defined terms based on random vector representation. This is convenient especially when we do matrix-vector operations during estimation. Another notation for the definition of these terms is the sample-based notation. For example, auto-correlation function can be written as,

$$R_x[m] = E\{x[n]x[n-m]^*\}$$

Similarly, cross-correlation and cross-covariance functions can be written as,

$$R_{xy}[m] = E\{x[n]y[n-m]^*\} = C_{xy}[m] + m_x m_y^*$$

$$C_{xy}[m] = E\{(x[n]-m_x)(y[n-m]-m_y)^*\}$$

Properties of Autocorrelation Function

There are some properties of autocorrelation function which are used in detection and estimation problems.

a) $R_x[0] = E\{x[n]x^*[n]\} = P_{avg} \geq 0$
b) $|R_x[n]| \leq R_x[0]$
c) $R_x[n] = R_x[-n]$

Above definitions for statistical terms are theoretical and cannot be computed perfectly in practical applications. In practice, we need to use “sample” based statistical terms which are only estimates of theoretical terms. Therefore depending on the accuracy of these estimates, our inference performance changes.

Consider sample correlation function

$$\hat{R}_x[l] = \frac{1}{N_s}\sum_{n=0}^{N_s-l-1} x[n+l]x^*[n] = \frac{1}{N_s}\sum_{n=l}^{N_s-1} x[n]x^*[n-l] \qquad 0 \leq l < N_s$$

Similarly, cross-correlation function can be written as,

$$\hat{R}_{xy}[l] = \frac{1}{N_s}\sum_{n=0}^{N_s-l-1} x[n+l]y^*[n] = \frac{1}{N_s}\sum_{n=l}^{N_s-1} x[n]y^*[n-l]$$

Deterministic Correlation and Its Relation to Convolution

In the previous part, we have defined the theoretical and sample definitions for cross-correlation. More specifically the sample cross-correlation is written as,

$$\hat{R}_{xy}[l] = \frac{1}{N_s}\sum_{n=0}^{N_s-l-1} x[n+l]y^*[n] = \frac{1}{N_s}\sum_{n=l}^{N_s-1} x[n]y^*[n-l]$$

We would like to relate this expression with convolution. Consider the convolution of two N-point sequences, (0≤ $n < N$)

$$x[n] * y^*[-n] = \sum_{m=0}^{N-1} x[m]y^*[-(n-m)] = \sum_{m=n}^{N-1} x[m]y^*[m-n]$$

Note that y[n] is also a sequence which is nonzero in 0≤n<N-1.

Therefore cross-correlation can be written in terms of convolution as,

$$\hat{R}_{xy}[l] = \frac{1}{N_s}x[n] * y^*[-n]$$

The DTFT of the above expression is the power spectral density function, and it is given as,

$$\hat{S}_{xy}(e^{j\omega}) = \frac{1}{N_s}X(e^{j\omega})Y^*(e^{j\omega})$$

Ergodic Random Process: If a random process is stationary and signal average is equal to the ensemble average with probability one, then that random process is called as ergodic. An ergodic random process should be stationary. However not every stationary random process is ergodic.

Mean Ergodic:

$$E\{x(t)\} = \int_{-\infty}^{\infty} x f_x(x)dx = \lim_{T\to\infty}\frac{1}{2T}\int_{-T}^{T} x(t)dt$$

Correlation Ergodic:

$$R_{xx}(\tau) = E\{x(t)x(t-\tau)\} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x_1 x_2 f_{x(t),x(t-\tau)}(x_1,x_2)dx_1dx_2 = \lim_{T\to\infty}\frac{1}{2T}\int_{-T}^{T} x(t)x(t-\tau)dt$$

Above approach can be used to find other results including ergodicity in cross-correlation.

Detection of Signals in Noise: Matched Filter

In radar, sonar and communications, the detection of the presence of a signal in noise and its time-of-arrival are desired. Matched filter is the optimum filter for maximizing the signal-to-noise ratio (SNR) for detecting signals in additive noise. Consider a desired signal s[n] corrupted by noise v[n].

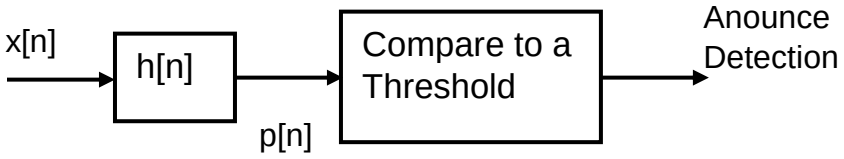
$$x[n] = s[n] + v[n], \qquad 0 \leq n < N$$

where s[n] is a deterministic signal that is transmitted and v[n] is the noise. There are two cases in the received signal, x[n], i.e.

$$x[n] = \begin{cases} v[n], & no\ signal \\ s[n] + v[n], & signal\ present \end{cases}$$

If v[n] is zero-mean white noise (noise samples are uncorrelated) the optimum way to detect the presence of the signal is the correlation detector or matched filter. The optimality criteria is the SNR maximization. Hence this detector has the best performance in SNR within all possible detectors.

The following block diagram summarizes the matched filter operation. The received signal is filtered by h[n] and compared to a threshold to determine the signal presence as well as the time-of-arrival (TOA). If the filter h[n] is selected as the time-reversed transmitted signal, convolution corresponds to correlation function of s[n].



h[n] is the matched filter and it is given as,

$$h[n] = s[N-1-n], \;\; 0 \leq n < N$$

The matched filter output is,

$$p[n] = \sum_{k=0}^{N-1} x[k]h[n-k]$$

If we evaluate p[n] at n=N-1,

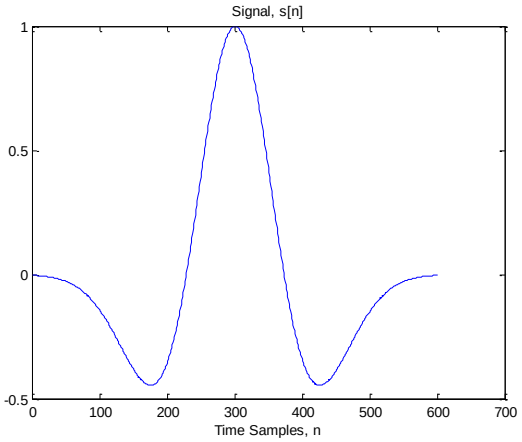
$$p[N-1] = \sum_{k=0}^{N-1} x[k]s[k] = \hat{R}_s[0]$$

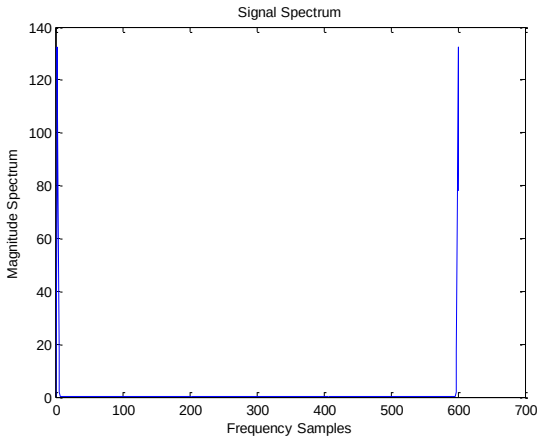
Hence we see the largest value at n=N-1 if the signal s[n] exists in the received signal.

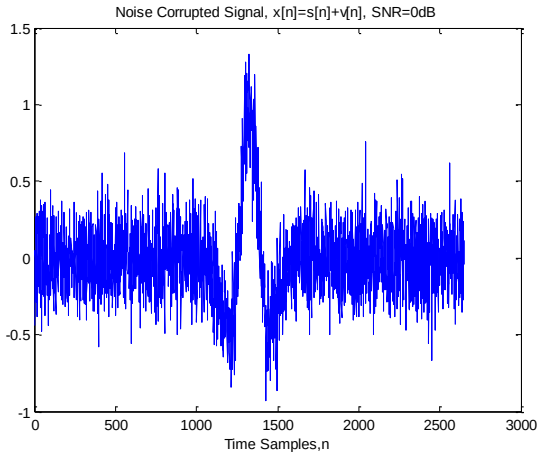
Note that in frequency domain,

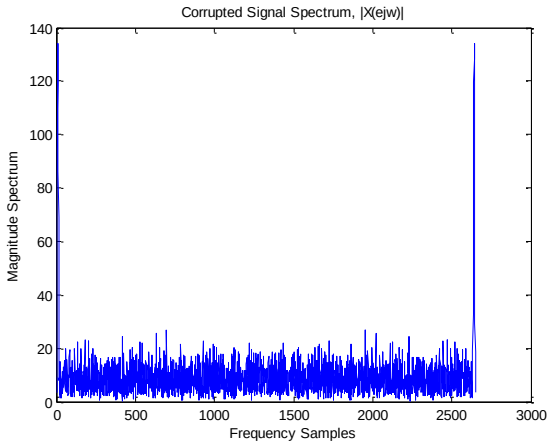
$$P(e^{j\omega}) = X(e^{j\omega})S^*(e^{j\omega})e^{-j\omega(N-1)} = |S(e^{j\omega})|^2 e^{-j\omega(N-1)} + V(e^{j\omega})S^*(e^{j\omega})e^{-j\omega(N-1)}$$

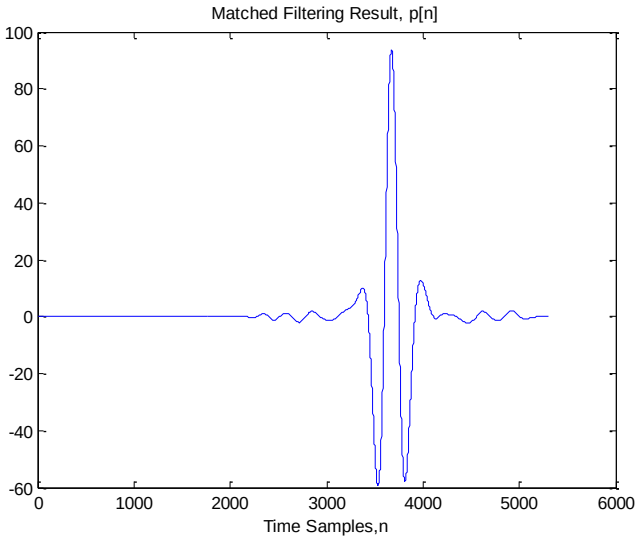
In the following Figures, the application of matched filter and its results are shown.









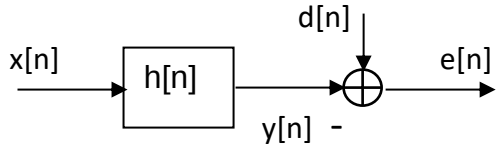

Matched Filtering Result, p[n]
100
80
60
40
20
0
-20
-40
-60
0
1000
2000
3000
4000
5000
6000
Time Samples,n

5.1. Optimum Filtering: Wiener Filters

The most common type of filters encountered in practice are deterministic filters in the form of a lowpass, highpass, etc. While these filters are easily designed and used, they are far from being optimum especially due to fact that they do not use the information present in signal statistics. Optimum filters perform the best with respect to a certain optimality criteria and they are designed using the signal statistics. While there are several other optimality criteria, we concentrate on a special one, namely the mean-square-error, MSE. MSE is the most frequently used optimality criteria since it has a simple mathematical formula and closed form results can be conveniently obtained. Furthermore, it results practically good filters. In the following part, the theoretical background for MSE optimum filters or Wiener Filters is presented.

5.2. Derivation of Wiener Filters

The following figure shows the case for the design of a Wiener filter.



The problem for MSE optimum filters can be outlined as follows. Given the input sequence, x[n], desired response, d[n] and the form of the filter structure, i.e.,

$$y[n]=\sum_{k=0}^{\infty}h[k]x[n-k],\ \ n=0,1,... \qquad (1)$$

the target is to find filter coefficients, h[k], such that the MSE is minimized. Note that, y[n] is the Wiener filter output. While it is possible to have IIR Wiener filters, FIR Wiener filters are considered in this note for simplicity. The definitions for the error and MSE are given as follows, i.e.,

$$\begin{aligned} &e[n]=d[n]-y[n]=d[n]-\sum_{k=0}^{\infty}h[k]x[n-k] \\ &MSE=J=E\left\{e[n]^2\right\} \end{aligned} \qquad (2)$$

where E{} is the expectation operator and d[n] is the desired signal. Note that there may be limited information about the desired signal. In Wiener filtering, second order statistics for d[n] is used to obtain the filter coefficients. In order to find the optimum coefficients, derivative of J with respect to h[k] should be equated to zero, i.e.,

$$\nabla_k J=\frac{\partial J}{\partial h_k}=2E\left\{e[n]\frac{\partial e[n]}{\partial h_k}\right\}=-2E\{e[n]x[n-k]\}=0 \qquad (3)$$

Above is the principle of orthogonality. In other words, J attains its minimum if-and-only-if (iff) the estimation error e[n] is orthogonal to the input, x[n]. If we insert, e[n] into (3), we obtain,

$$E\left\{\left(d[n]-\sum_{i=0}^{\infty}h[i]x[n-i]\right)x[n-k]\right\}=0$$

$$\sum_{i=0}^{\infty}h[i]E\{x[n-i]x[n-k]\}=E\{d[n]x[n-k]\} \quad (4)$$

$$\sum_{i=0}^{\infty}h[i]R_x[i-k]=r_{dx}[k], \quad k=0,1,2,....$$

In the above equation, $\mathbf{R_x}$ is the autocorrelation matrix of the input, and $\mathbf{r_{dx}}$ is the cross-correlation vector of the desired and input signals. The last equation in (4) is called as the Wiener-Hopf equation and can be more compactly written in terms of matrix-vector notation as,

$$\mathbf{R_x h}=\mathbf{r_{dx}} \quad (5)$$

The minimum MSE for the Wiener filtering is given as,

$$MSE_o=\sigma_e^2=R_d[0]-\mathbf{h}^{\mathrm{T}}\mathbf{r_{dx}}=R_d[0]-\sum_{m=0}^{P-1}h[m]r_{dx}[m] \quad (6)$$

where $R_d[0]$ is the 0th term of the autocorrelation sequence of the desired signal, and P is the length of the FIR Wiener filter. Hence (6) can be used to compute theoretical minimum MSE (even without computing y[n]). It is also possible to compute least-squares-error (LSE) which is a deterministic counterpart of MSE and it is computed by using signal samples, i.e.,

$$LSE=\frac{1}{N}\sum_{n=0}^{N-1}|e[n]|^2 \quad (7)$$

MSE in (6) is theoretical, LSE in (7) is the practical error. As the number of samples, N, increases, LSE approaches to MSE under ideal conditions.

5.2. Applications of Wiener Filters

Wiener filters can be used in a variety of applications. Filtering of a signal in noise (or noise removal), prediction of a signal in noise, smoothing of a signal in noise, and linear prediction are some examples of these applications. In this experiment, two applications namely the filtering of a signal in noise and prediction are considered. In the following part, examples of such applications are given. However, first a signal with a known correlation function is discussed.

5.2.1. Innovations Process

Innovation is the difference between the observed value of a variable at time n, and the optimal prediction of that value based on the previous samples. Hence if the prediction is accurate, then the innovations samples are uncorrelated with each other, namely the white noise samples.

A regular process is obtained as the output of a filter when the filter input is white Gaussian noise.

Fig.1.a Generation of a regular process, s[n]. b)Innovations representation of the process in a. H(z) is a minimum-phase filter.

Let H(z) is selected as,

$$H(z)=\frac{1}{1-0.2z^{-1}} \qquad (8)$$

Then it is possible to show that the autocorrelation sequence for s[n] is given as,

$$R_s[m] = R_v[m] * h[m] * h^*[-m] = \frac{\sigma_v^2}{1-(0.2)^2}(0.2)^{|m|} = \frac{\sigma_v^2}{0.96}(0.2)^{|m|} \qquad (9)$$

where it is assumes that noise correlation function is $R_v[m] = \sigma_v^2\delta[m]$.

If we look at the z-transform of $R_s[m]$, we obtain complex power spectral density as,

$$S_s(z) = S_v(z)H(z)H^*(1/z^*) \qquad (10)$$

where H(z) is a LTI minimum-phase filter, $S_v(z)$ is the z-transform of the correlation function of noise, $R_v[m]$. $S_s(z)$ is called as the complex power spectral density function. If we find the complex power spectral density function for Fig1.b at the output, we can easily show that it corresponds to white noise.

By definition, any process that is regular has an innovations representation. The operation in Fig 1.b is also seen as the whitening procedure.

In certain applications, methods are applicable only for white noise. When the noise is colored, a whitening process should be applied similar to Fig1.b.

Ex: **Whitening Filter**

Assume that power spectral density for a random process is given as,

$$S_x(e^{j\omega}) = \frac{5}{4} - cos(\omega)$$

Above can be written as,

$$S_x(e^{j\omega}) = \left(1 - \frac{1}{2}e^{-j\omega}\right)\left(1 - \frac{1}{2}e^{j\omega}\right)$$

$$S_x(z) = \left(1 - \frac{1}{2}z^{-1}\right)\left(1 - \frac{1}{2}z\right)$$

In order to whiten x[n], we need a minimum-phase filter obtained from $S_x(z)$, i.e.

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

Hence H(z) is the whitening filter for $S_x(z)$.

Power Spectral Density function is the Discrete-Time Fourier transform of the correlation function, i.e.

$$S_s\left(e^{j\omega}\right) = S_s(z)\Big|_{z=e^{j\omega}} = \sum_{n=-\infty}^{\infty} R_s(n)e^{-j\omega n} \quad (11)$$

$S_s(e^{j\omega})$ is always real and positive (prove). Above is the theoretical definition of the Power Spectral Density function. In practice, under limited information about the observed signal, we use a deterministic estimate of the power spectral density, (which is also called as the periodogram estimate) i.e.

$$\hat{S}_s\left(e^{j\omega}\right) = \frac{1}{N}\left|S\left(e^{j\omega}\right)\right|^2 \quad (12)$$

where $S(e^{j\omega})$ is the DTFT of s[n] signal, and N is the length of the signal. This shows the value of the Fourier Transform magnitude square in the sense that it is the deterministic counterpart of the Power Spectral Density function. Note that above result in (12) can be easily obtained from the deterministic correlation function definition for a causal sequence s[n], i.e.

$$\hat{R}_s[m] = \frac{1}{N}\sum_{n=0}^{N-1} s[n]s^*[n-m] = \frac{1}{N}s[m] * s^*[-m] \quad (13)$$

5.2.2. Filtering of a signal in noise

Assume that the signal of interest, s[n], is corrupted by white Gaussian sequence v[n] where $R_v[m]=\sigma_v^2\delta[m]$. Therefore observed signal, x[n], is given as,

$$x[n]=s[n]+v[n] \qquad (14)$$

Noise is uncorrelated with the signal s[n].

In this case, desired signal is d[n]=s[n] and $R_d[m]=R_s[m]$. Note that the problem is as follows. Given the noisy signal x[n], its correlation function, $R_x[m]$ and the cross-correlation $R_{dx}[m]$, find the MSE optimum filter coefficients h[n] such that the filter output is the desired signal. The desired filter length is given as P=3. It is possible to write $R_{dx}[m]$ and $R_x[m]$ as follows, i.e.,

$$R_{dx}[m]=E\{s[n]x[n-m]\}=E\{s[n](s[n-m]+v[n-m])\}=R_s[m]=\frac{1}{0.96}(0.2)^{|m|} \qquad (15)$$

$$R_x[m]=E\{(s[n]+v[n])(s[n-m]+v[n-m])\}=R_s[m]+R_v[m]=\frac{1}{0.96}(0.2)^{|m|}+\sigma_v^2\delta[m] \quad (16)$$

Note that v[n] and s[n] are assumed to be uncorrelated. Using the Wiener-Hopf equations in (4) or (5), we obtain,

$$\begin{bmatrix} R_x[0] & R_x[-1] & R_x[-2] \\ R_x[1] & R_x[0] & R_x[-1] \\ R_x[2] & R_x[1] & R_x[0] \end{bmatrix}\begin{bmatrix} h[0] \\ h[1] \\ h[2] \end{bmatrix}=\begin{bmatrix} R_{dx}[0] \\ R_{dx}[1] \\ R_{dx}[2] \end{bmatrix} \qquad (17)$$

Note that correlation matrix is Hermitian symmetric, i.e., $R_x[-1]=R_x^*[1]$. Once the numerical values for the above expression are used, Wiener filter coefficients can be easily obtained through matrix inversion, i.e., $\mathbf{h_{opt}}=\mathbf{R_x^{-1}r_{dx}}$. Theoretical MSE is obtained from (6).

Example: Let the length of the Wiener filter be P=2 and $\sigma_v^2=1$. Using (15),(16), and (17)

$$R_x=\begin{bmatrix} 2.0417 & 0.2083 \\ 0.2083 & 2.0417 \end{bmatrix},\quad R_{dx}=\begin{bmatrix} 1.0417 \\ 0.2083 \end{bmatrix}$$

are obtained. Then h is found as

$$h=[0.5051\ 0.0505]^T$$

and MSE becomes, $MSE=R_d[0]-h^TR_{dx}=0.50506$. If the filter length is increased to P=3, $h=[0.5050\ 0.05\ \ 0.005]^T$ and MSE is decreased to MSE=0.5050. Hence there is a small improvement in MSE as the length is increased. Note that this is not always the case and it depends on the problem and the signal statistics.

5.2.3. Prediction of a signal in noise

In this case, the problem is to predict the signal samples before K-steps. Hence the desired signal is d[n]=s[n+K] where K=2 is selected for simplicity. $R_x[m]$ does not change and it is the same as in (14). $R_{dx}[m]$ changes and it is given below,

$$R_{dx}[m] = E\{s[n+2]x[n-m]\} = E\{s[n+2](s[n-m]+v[n-m])\} = R_s[m+2] = \frac{1}{0.96}(0.2)^{|m+2|} \quad (18)$$

It is possible to write a similar expression to (12) in this case and find the Wiener filter coefficients. MSE in this case is

$$MSE = R_d[0] - \sum_{m=0}^{2} h[m]\frac{1}{0.96}(0.2)^{|m+2|} \quad (19)$$

5.3. Comparison of Wiener Filter and Matched Filter

Matched Filter (MF) and Wiener Filter (WF) are two different filters since they use two different optimization criteria.

Matched Filter uses the SNR maximization, i.e.

$$SNR = \frac{|h^T s|^2}{E\{|h^T v|^2\}}$$

where h is the Matched filter vector, s is the signal and v is the noise.

Wiener Filter uses the minimization of MSE, i.e.

$MSE = E\{|h^H x - s|^2\}$

MF maximizes the SNR possibly at a specific instant of time. WF minimizes the MSE for all signal samples. In the ideal case, SNR is approximately proportional to 1/MSE.

6. ADAPTIVE FILTERING

6.1. Basics of Adaptive Filtering

If the signal statistics does not change in time and there is sufficient information, optimum solution (in MSE sense) for a variety of problems can be obtained by solving the Wiener-Hopf equation, i.e.,

$$\mathbf{h}_{\mathbf{opt}} = \mathbf{R}_{\mathbf{x}}^{-1}\mathbf{R}_{dx}$$

where $\mathbf{R}_x$ is the input signal correlation matrix, $\mathbf{R}_{dx}$ is the cross-correlation between the input and the desired signal.

Above is possible if we have a block of signal to compute $\hat{R}_x, \hat{R}_{dx}$ from the samples. Hence a blockwise processing is usually performed in practice.

For many real-world problems, signal statistics change in time and adaptive filtering should be used in order to adopt to the input signal statistics. There are two approaches for adaptive filtering,

a) Sample Adaptive Processing
b) Block Adaptive Processing

In this course, we will consider a very simple form of adaptive algorithm, namely the Least-mean Square, LMS, algorithm. While LMS algorithm is very simple in its form, it has found a wide spread application in signal processing due to its computational efficiency and effectiveness. Figure 1 shows the block diagram of the Adaptive filter.

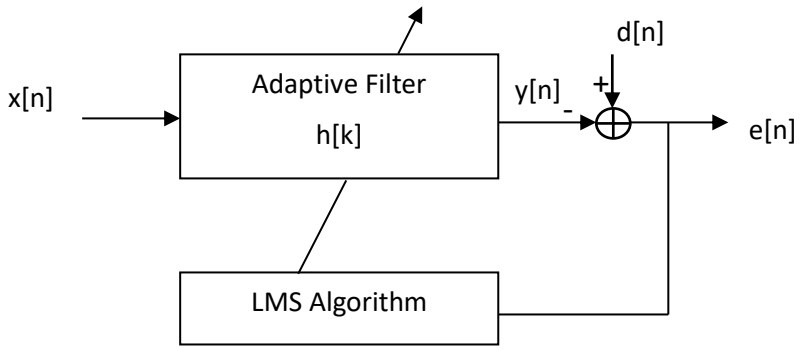


Figure 1. Block diagram of an adaptive filter structure

In Figure 1, x[n] is the input signal, y[n] is the filter output, d[n] is the desired signal, e[n] is the error signal, and h[k] is the coefficient of the adaptive filter.

There are different forms of adaptive filters such as IIR adaptive filters, FIR adaptive filters. We will mainly concentrate on the FIR adaptive filter for simplicity. Note that FIR adaptive filters are always stable and they do not require a check for stability as the filter coefficients change in time.

In Figure 2, a detailed structure of the FIR adaptive filter is given.

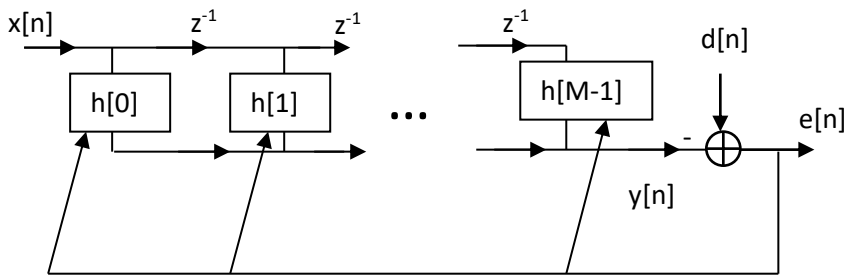


Figure 2. Detailed structure of an FIR (tapped delay line) adaptive filter structure.

LMS algorithm is used to update the filter coefficients at each iteration. The coefficient update equation is given as,

$$\begin{aligned} &Sample\ Form: h_k[n+1] = h_k[n] + \mu e^*[n]x[n-k],\ k = 0,1,...,M-1 \\ &Vector\ Form: \ \mathbf{h}[n+1] = \mathbf{h}[n] + \mu e^*[n]\mathbf{x}[n] \end{aligned} \quad (1)$$

In equation (1), it is assumed that there are M adaptive filter coefficients and n is the iteration or input sample index.

6.2.Derivation of the LMS Algorithm

LMS algorithm is used for the update of the adaptive filter coefficients at each iteration. The derivation of the algorithm is based on the minimization of the error between the desired response d[n] and the adaptive filter output, y[n].

LMS is a steepest descent type (also known as the gradient descent type) approach. Figure 3 shows how the error at each step is decreased in the gradient of the error direction.

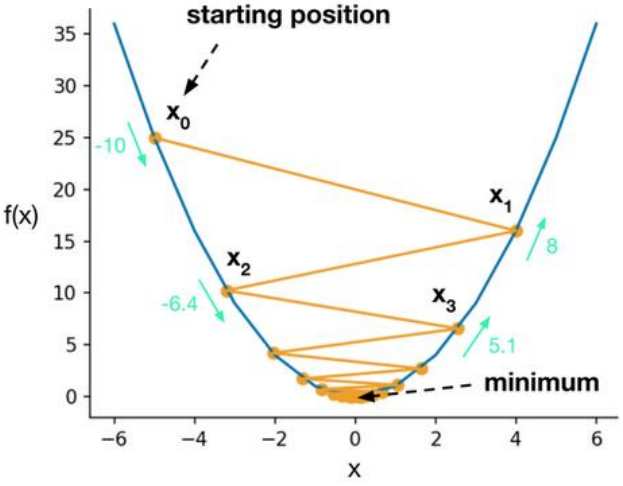


Figure 3. Gradient descent characteristics of the LMS algorithm on the error surface.

Let the error function, C[n], be expressed as,

$$C[n] = E\{|e[n]|^2\} \qquad (2)$$

For the application of steepest descent, we take the partial derivative of C[n] w.r.t. filter vector **h**. The error, e[n] is defined as,

$$e[n] = d[n] - \boldsymbol{h}^H \boldsymbol{x}[n] \qquad (3)$$

$$\nabla_{h^H} e[n] = -\boldsymbol{x}[n] \qquad (4)$$

If we take the gradient of C[n],

$$\nabla_{h^H} C[n] = \nabla_{h^H} E\{e[n]e[n]^*\} = -E\{\boldsymbol{x}[n]e[n]^*\} \qquad (5)$$

$\nabla_{h^H} C[n]$ is a vector that points towards the steepest ascent of the cost function. To find the minimum of the cost function, we need to take a step in the opposite direction. Hence we update the filter vector at the nth iteration as,

$$\boldsymbol{h}[n+1] = \boldsymbol{h}[n] - \mu \nabla_{h^H} C[n] = \boldsymbol{h}[n] + \mu\, E\{\boldsymbol{x}[n]e[n]^*\} \quad (6)$$

Above is a vector form of the update equation. Here, μ is the step size which determines how fast we move on the error surface.

Usually expectation operation, E{} , in (6) is replaced by the sample estimator. An unbiased estimator in this case becomes,

$$E\{\boldsymbol{x}[n]e[n]^*\} = \frac{1}{N}\sum_{i=0}^{N-1} \boldsymbol{x}[n-i]e[n-i]^* \qquad (7)$$

Note that with the above simplification, LMS algorithm loses the optimality and hence it is a suboptimal algorithm. We take N=1 for simplicity, then the final LMS update equation in vector form becomes,

$$\boldsymbol{h}[n+1] = \boldsymbol{h}[n] + \mu e[n]^* \boldsymbol{x}[n] \qquad (8)$$

Usually, step size is selected as a small value. If it is very small, update is slow and the adaptive filter cannot adopt to the changes of the signal statistics quickly. But the final LSE at the output becomes small. If it is large, then the adaptation is quick but the LSE is large. The range of values for the step size is given below,

$$0 < \mu < \frac{2}{\lambda_{max}} \qquad (9)$$

where λ_{max} is the largest eigenvalue of the correlation matrix of the input, $\mathbf{R}_x$. Note that if the step size is large, adaptive filter may be unstable and the filter output becomes very large and saturated. A value for maximum convergence speed can be given as below,

$$\mu = \frac{2}{\lambda_{max} + \lambda_{min}} \qquad (10)$$

6. ADAPTIVE FILTERING Part 2

6.3. Excess Mean-Squared Error

In adaptive filter, we can assume that the adaptive filter vector is a perturbed version of the optimum filter, i.e.

$$\boldsymbol{h} = \boldsymbol{h}_{opt} + \boldsymbol{h}_{mis} \quad (1)$$

where $\mathbf{h}_{opt}$ is the optimum filter and $\mathbf{h}_{mis}$ is the misadjustment. We would like to find out the contribution of each term to the final MSE. Using the definition of error,

$$e[n] = d[n] - \boldsymbol{h}^H \boldsymbol{x}[n] \quad (2)$$

$$e[n] = d[n] - \boldsymbol{h}_{opt}^H \boldsymbol{x}[n] - \boldsymbol{h}_{mis}^H \boldsymbol{x}[n] \quad (3)$$

$$e[n] = e_0[n] - \boldsymbol{h}_{mis}^H \boldsymbol{x}[n] \quad (4)$$

In (4) $e_0[n]$ is the error which will be obtained in case of optimum Wiener solution. If we compute the MSE from (4),

$$C[n] = E\{|e[n]|^2\} = C_{min} + \boldsymbol{h}_{mis}^H \boldsymbol{R}_x \boldsymbol{h}_{mis} \quad (5)$$

In (5), C_{min} is the MSE for the Wiener solution and the second term, C_{ex}, is the excess MSE due to misadjustment. It is know that

$$\lambda_{min} \leq C_{ex} \leq \lambda_{max}$$

In other words, excess MSE is a value between the minimum and maximum eigenvalues of the input correlation matrix $\mathbf{R}_x$.

An extension of the above analysis is the discussion on the convergence speed for the LMS algorithm based on the eigenvalue spread of Rx. A faster convergence is achieved if λ_{max} is close to λ_{min}. Eigenvalue spread can be defined as,

$$\gamma = \frac{\lambda_{max}}{\lambda_{min}} \geq 1 \quad (6)$$

Hence as the eigenvalue spread increases, convergence speed decreases.

6.4 Variants of the LMS Algorithm

6.4.1. Normalized LMS (NLMS) Algorithm

When the input signal is large, standard LMS algorithm experiences gradient noise amplification. It is possible to overcome this problem by normalizing the update equation with the input norm, i.e.

$$\boldsymbol{h}[n+1] = \boldsymbol{h}[n] + \frac{\mu}{a + \|x[n]\|^2} e[n]^* \boldsymbol{x}[n] \qquad (7)$$

where a>0 is a small scalar to avoid division by zero.

NLMS has data dependent adaptation step size. It offers a better trade-off in terms of convergence speed and excess error. However it has higher computational cost than the LMS algorithm.

6.4.2. Sign LMS Algorithm

In high bitrate telecommunication systems, time is critical and low computational complexity has a higher priority. In this case, different versions of sign LMS algorithm is used. Note that sign operation is defined as,

$$sgn(a) = \begin{cases} 1; & a > 0 \\ 0; & a = 0 \\ -1; & a < 0 \end{cases}$$

Note that the use of sign LMS algorithm leads to larger excess MSE and convergence rate decreases. In return, we have faster computation. The variants of sign LMS algorithm are presented below.

a) Sign LMS Algorithm

$$\boldsymbol{h}[n+1] = \boldsymbol{h}[n] + \mu\, sgn(e[n])\boldsymbol{x}[n] \qquad (8)$$

b) Clipped LMS Algorithm

$$\boldsymbol{h}[n+1] = \boldsymbol{h}[n] + \mu\, sgn(\boldsymbol{x}[n])e[n] \quad (9)$$

c)Zero Forcing LMS Algorithm (Sign-Sign LMS)

$$\boldsymbol{h}[n+1] = \boldsymbol{h}[n] + \mu\, sgn(\boldsymbol{x}[n])sgn(e[n]) \quad (10)$$

6.5. Applications of Adaptive Filters

Adaptive filters are used in radar, sonar, communications, signal processing, biomedical engineering, etc. In this part, we will only consider a small subset of all applications.

6.5.1 System Identification

In this application, we would like to find the impulse response of an unknown linear system. Note that the impulse response of this linear system may change in time and our adaptive filter is intended to track the coefficients. The key in this case is the use of same input for both adaptive filter and the unknown system. The structure for this problem is given below.

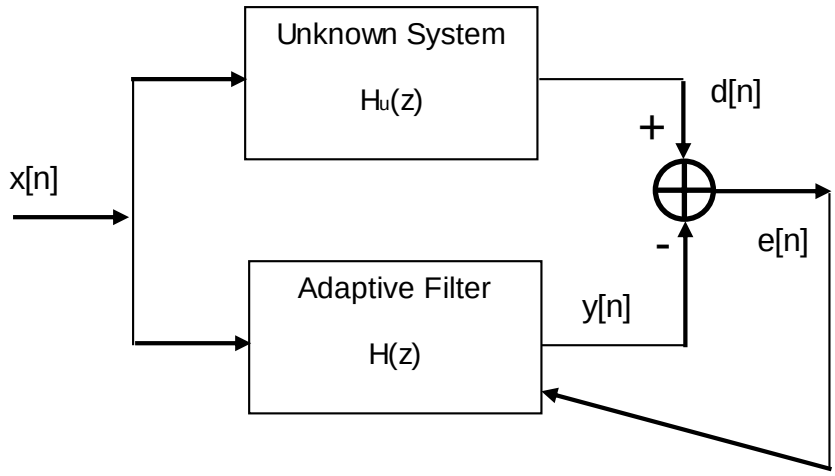


Figure 1. Adaptive filter structure for system identification.

Note that it is possible to have noise or interference at the output of the unknown system. In this case, we ignore it for simplicity. Under this case, h[n] converges to $h_u[n]$ in time.

6.5.2. Single-Input Multi-Output (SIMO) System Identification

In certain applications including communications, we need to find multiple unknown systems at different channels of the structure. Figure 2 shows a two-channel SIMO system identification problem. In this case, $h_1[n]$ and $h_2[n]$ are the unknown linear filters whereas $g_1[n]$ and $g_2[n]$ are the adaptive filters. When this system converges, $g_1[n]=h_2[n]$ and $g_2[n]=h_1[n]$.

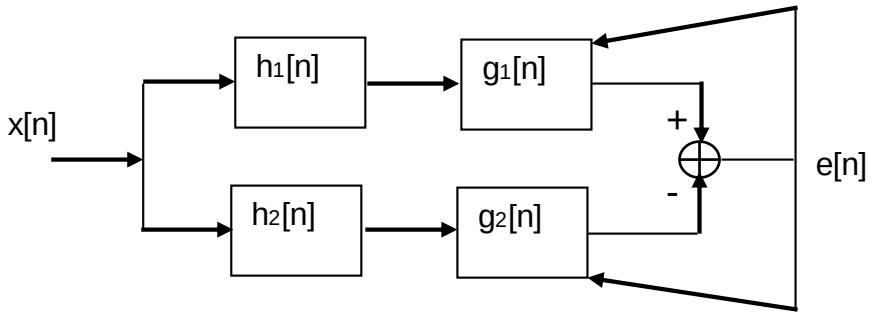


Figure 2. Single-input Multi-output system identification.

6.5.3. Interference and Noise Cancellation

One of the main features of adaptive filters is their ability to adopt to the time varying signal characteristics. One application which uses this feature is the interference cancellation as shown in Figure 3.

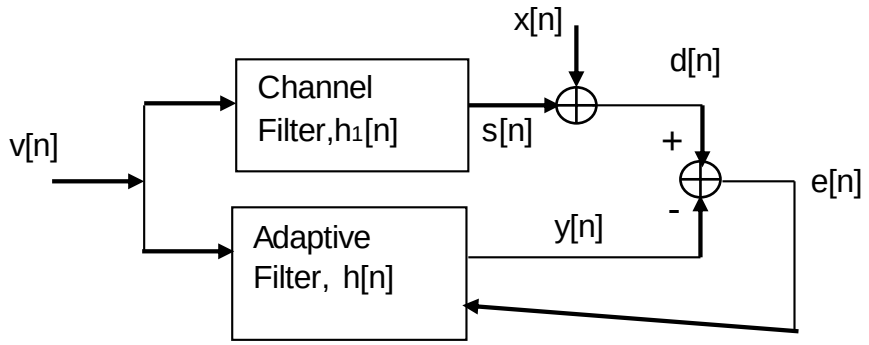


Figure 3. Interference cancellation by adaptive filtering.

In this system, v[n] is a noise sequence which is common to both channel filter and the adaptive filter. x[n] is the signal of interest which we would like to obtain at the output of this structure. Note that we assume that there is no correlation between x[n] and v[n]. Under this condition, adaptive filter output, y[n], converges to s[n]. Hence e[n] approaches to x[n] in time.

6.5.4.Inverse Filtering

One of the problems in signal processing is to obtain the inverse of a LTI filter. This problem is solved in different ways and we have a solid theoretical foundation when the filter is time-invariant. The system structure in Figure 4 can be used for both time-invariant and time-varying cases.

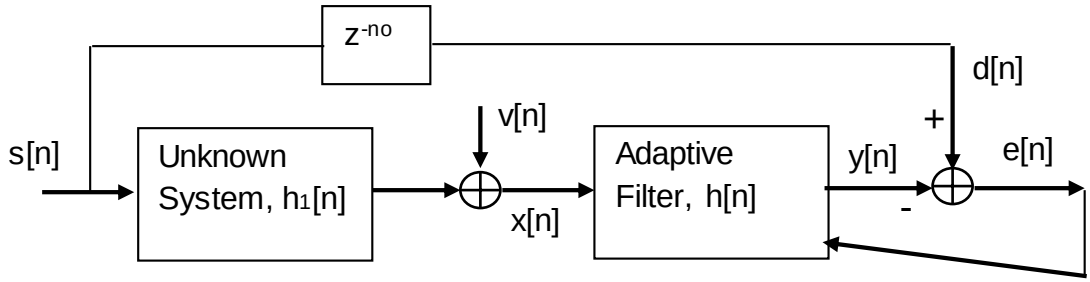


Figure 4. Use of adaptive filter for inverse filter design.

In this case, let us assume that v[n]=0 for simplicity. Adaptive filter will converge to the inverse of $h_1[n]$ so that its output is close to a delayed input, i.e. $y[n]=s[n-n_o]$. If v[n] is not zero, the quality of the final inverse filter will decrease depending on the statistics and amplitude of noise, v[n].

Wiener Inverse Filter

$$H_{inv}[k] = \frac{|H[k]|^2}{|H[k]|^2 H[k] + \frac{1}{SNR[k]}}$$

7. Image and Video Processing

An image is a two dimensional signal where x and y coordinates correspond to the location of information. Usually x and y are positive integer indices for pixel values and image is treated as a grid of discrete elements.

Video has an additional axis, namely time. Hence video is the time sequence of images. It has three dimensions. While video and image are seen as different types of signals in general, video processing involves image processing performed at each frame.

Image and video processing are computationally expensive compared to 1D processing of signals. Usually efficient algorithms and simple filters are preferred in order to decrease the computational complexity. While 1D signals can be processed in standard hardware environments such as microprocessors, image and video signals usually require special devices. Two common hardware for this purpose are FPGA and GPU. FPGA is usually the choice for low level applications such as controlling leds, or grid of leds. On the other hand, GPU is used for high level image and video processing.

In both FPGA and GPU processing, there are two components that should be taken into consideration, namely the hardware and software. In other words, having a very top notch FPGA or GPU does not mean that a sophisticated processing can be implemented easily. One key component for this end is the “software” which should be easily accessible and programmable. Unfortunately, both FPGA and GPU software environments require considerable expertise. In addition, this expertise should be renewed for every three to five years due to the advances in technology.

In image and video processing, we can divide the processing tasks as simple and high level tasks.

Simple Image Processing Tasks:

a. Image filtering
b. Edge detection
c. 2D Fourier Transform
d. Inverse Filtering
e. Thresholding
f. Color Conversion

High Level Image Processing Tasks:

a. Face detection and recognition
b. Object recognition and classification
c. Machine vision
d. Virtual reality

Another example of high level tasks is the license plate recognition using Deep Learning frameworks. Above examples show the high level complexity and variety of different applications in image and video processing.

7.1. Image Representation

Image representation or format is important and should be known well to understand the processing involved with images. There are several different formats for image representation. Some of these formats are explained below:

a.Gray-scale images: These images have only one color dimension. They have grayscale for representing the information. An example grayscale image has 480x640 dimensions and each pixel value is an integer between 0-255. It is also possible to use floating point values for pixel values but the common format is unsigned integer representation.

b. Red-Green-Blue(RGB) images: These images have three fields for color representation, namely red, green and blue respectively. An example of a RGB image has 480x640 pixels for each color component. In other words, its dimension is 480x640x3.

c.YUV images: This is another type of image format. Y, U, and V correspond to luminance, chroma1 and chroma2 respectively. Another version of this format is YCbCr (luminance, blue difference chroma, red difference chroma). An example for this type of image has 480x640x3 dimension.

Note that the computational complexity of processing color images is three times more than the grayscale images. This is the reason for conversion from RGB to Grayscale before processing in many cases.

7.2. Image Filtering

In image filtering, there are two common approaches, namely linear filtering and nonlinear filtering respectively. While in one dimensional signal processing, nonlinear filters are seldom used, they are more frequently used in image processing. One such nonlinear filtering example is linear filtering first and then thresholding to obtain binary images for finding edges.

Scaling is an important part of filtering to avoid clipping in image filtering. Consider a 3x3 2D smoothing (lowpass) filter below,

$$h[n_1, n_2] = \begin{bmatrix} 0 & \frac{1}{5} & 0 \\ \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ 0 & \frac{1}{5} & 0 \end{bmatrix}$$

As it is seen from this filter, the sum of the coefficients is one. Therefore if an image with pixel values of 255 is filtered, the result is not greater than the unsigned 8 bit representation and hence there is no clipping. Note that clipping is a kind of information loss even though it is a mild case of such defects.

7.2.1 Circular Convolution

In 1D Signal Processing, linear convolution is usually preferred. Note that when two signals, each with length L_1 and L_2 are linearly convolved, the result is a sequence with length L_1+L_2-1. If you want to implement convolution in Fourier domain, then the $N=L_1+L_2-1$ point DFT of these sequences should be used.

In image processing, given an image of size 256x256, we do not want to have any increase in dimensions. Hence the result of the convolution or filtering is desired to be the same size. In this case, we obtain "circular convolution". Circular convolution can be seen as the aliased version of linear convolution. Consider an image of size 256x256 and a 3x3 filter. After circular convolution, the resulting image size is 256x256. The 254x254 part of this image is exactly the same as the result of linear convolution while the rest is aliased. A complete example for a comparison of linear and circular convolution is given below.

Example: Let x[n] be a L_1=4 point sequence and h[n] is a L_2=3 point sequence as below,

$$\{x[n]\}_{n=0}^{3} = [a \quad b \quad c \quad d]$$

$$\{h[n]\}_{n=0}^{2} = [1 \quad 2 \quad 1\,]$$

Linear convolution for these two sequence is,

$$\{y[n]\}_{n=0}^{5} = x[n] * h[n] = [a \quad 2a+b \quad a+2b+c \quad b+2c+d \quad c+2d \quad d]$$

Consider N=4 point circular convolution. It will be an aliased version of linear convolution and can be written as,

$$y_c[n] = \sum_{k=-\infty}^{\infty} y[n+kN], \qquad 0 \le n \le N-1$$

In this example, it is sufficient to consider only k=1 case,

$$y_c[n] = y[n] + y[n+4]$$

$$\{y_c[n]\}_{n=0}^{3} = [a+c+2d \qquad 2a+b+d \qquad a+2b+c \qquad b+2c+d]$$

As it is seen from the above example, first two samples (L_2-1) of y_c[n] are aliased, whereas the other samples are exactly the same as the linear convolution.

7.2.2. 2D Filtering Using 2D DFT

2D convolution between two sequences can be written as follows,

$$s[n,m] = x[n,m] * h[n,m]$$

$$s[n,m] = \sum_{k=0}^{L-1}\sum_{p=0}^{L-1} h[k,p]x[n-k,m-p], 0 \le n \le L+N-1,\ 0 \le m \le L+M-1 \quad (1)$$

where it is assumed that the image, x[n,m] is NxM and the filter is LxL. Convolution operation corresponds to multiplication in Fourier domain. Hence 2D DTFT is given as,

$$S(w_1,w_2) = X(w_1,w_2)H(w_1,w_2), \qquad -\pi \le w_1 \le \pi, \qquad -\pi \le w_2 \le \pi \ (2)$$

Note that in 2D Fourier domain there are two frequency axis, w_1 and w_2 respectively.

2D N_1xN_2 point DFT of h[n,m] can be written as,

$$H[k_1,k_2] = \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} h[n_1,n_2]e^{-j\frac{2\pi n_1 k_1}{N_1}}e^{-j\frac{2\pi n_2 k_2}{N_2}}, \quad 0 \le k_1 \le N_1, \qquad 0 \le k_2 \le N_2 \ (3)$$

An important observation is the fact that 2D DFT can be implemented as 1D DFT for the columns and 1D DFT for the rows, i.e.,

$$H[k_1,k_2] = DFT_{1D,n_2}\left\{DFT_{1D,n_1}\{h[n_1,n_2]\}\right\} \quad (4)$$

In the same token, 2D inverse DFT is written as,

$$h[n_1,n_2] = \frac{1}{N_1N_2}\sum_{k_1=0}^{N_1-1}\sum_{k_2=0}^{N_2-1} H[k_1,k_2]e^{j\frac{2\pi n_1 k_1}{N_1}}e^{j\frac{2\pi n_2 k_2}{N_2}}, \quad 0 \le n_1 \le N_1, \qquad 0 \le n_2 \le N_2 \quad (5)$$

$$h[n_1,n_2] = IDFT_{1D,k_2}\left\{IDFT_{1D,k_1}\{H[k_1,k_2]\}\right\} \quad (6)$$

If we consider the circular convolution and N_1=N, N_2=M, then the DFT for (1) can be written as

$$S[k_1,k_2] = X[k_1,k_2]H[k_1,k_2] \ \ 0 \le k_1 \le N_1, \qquad 0 \le k_2 \le N_2 \ (7)$$

where $X[k_1,k_2]$ and $H[k_1,k_2]$ are the N_1xN_2 DFT sequences. Then, time domain filtered image is obtained as,

$$s[n_1, n_2] = IDFT_{1D,k_2}\left\{IDFT_{1D,k_1}\{S[k_1, k_2]\}\right\} \quad 0 \le n_1 \le N_1, \qquad 0 \le n_2 \le N_2 \quad (8)$$

Note that the image $x[n_1, n_2]$ is $N_1 x N_2$ and the filtered image $s[n_1, n_2]$ is also $N_1 x N_2$.

7. Image and Video Processing: Part 2

Examples of 2D Convolution:

In this part, we will look at the examples of 2D linear and circular convolution. Consider 3x3 image $x[n_1,n_2]$ and 2x2 filter $h[n_1,n_2]$ given below,

$$x[n_1,n_2]=\begin{bmatrix}1 & 2 & 1\\ 2 & 3 & 2\\ 1 & 2 & 1\end{bmatrix},\quad h[n_1,n_2]=\begin{bmatrix}1 & -2\\ 2 & 1\end{bmatrix}$$

Linear convolution of these two sequences is given as,

$$y[n_1,n_2]=\begin{bmatrix}1 & 0 & -3 & -2\\ 4 & 4 & 0 & -3\\ 5 & 8 & 4 & 0\\ 2 & 5 & 4 & 1\end{bmatrix}$$

The circular convolution by 3x3 size is,

$$y_c[n_1,n_2]=\begin{bmatrix}2 & 5 & 1\\ 1 & 4 & 0\\ 5 & 8 & 4\end{bmatrix}$$

As it is seen from this example, the 2x2 samples in the diagonal are the same as the linear convolution.

7.2.3. Separable filters

There are two types of 2D FIR filter namely, separable and nonseparable filters. Separable filters can be decomposed in terms of single dimension, i.e.

$$h[n_1,n_2]=h[n_1]h[n_2]$$

Note that above is nothing but the multiplication of column and row vectors. Nonseparable filters cannot be decomposed as above. The advantage of separable filters is that they can be easily designed and implemented. However their performance may not be as good as the nonseparable filters in general. An example of separable filter is given below,

$$h[n_1,n_2]=\begin{bmatrix}1\\ 0\\ 1\end{bmatrix}\begin{bmatrix}1 & 0 & 1\end{bmatrix}=\begin{bmatrix}1 & 0 & 1\\ 0 & 0 & 0\\ 1 & 0 & 1\end{bmatrix}$$

As we have noted above, the design of separable 2D filters is simple. First a 1D filter with the desired specifications is designed using conventional tools. Then 2D filter with similar specifications is obtained as above.

It is easy to identify if a 2D filter is separable or not. If the rank of the filter matrix is one, then it is a separable filter.

7.3. 2D Filter Examples

There are several different 2D filters. We will present some examples of these that can be used for certain applications.

Example 1: Ideal Delay

Ideal delay in 2D case has the same effect similar to 1D case. The following filter shifts the image one sample in the x, and one sample in y dimension.

$$x[n_1, n_2] * \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} = x[n_1 - 1, n_2 - 1]$$

Example 2: Difference (Gradient) Filter

This filter is the discrete-time counterpart of derivative operator. Note that in 1D case, h[n]=δ[n]-δ[n-1] is the first order difference filter. Hence its counterpart which takes the difference in y dimension is given below

$$h[n_1, n_2] = \begin{bmatrix} 1 \\ -1 \end{bmatrix} [1 \quad 1] = \begin{bmatrix} 1 & 1 \\ -1 & -1 \end{bmatrix}$$

Difference operation can be applied in x dimension only by using,

$$h[n_1, n_2] = \begin{bmatrix} 1 \\ 1 \end{bmatrix} [1 \quad -1] = \begin{bmatrix} 1 & -1 \\ 1 & -1 \end{bmatrix}$$

A 3x3 version of the above filter is given below,

$$h[n_1, n_2] = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} [1 \quad 1 \quad 1] = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$$

This type of filters are used to find edges in x and y dimensions respectively in images.

Example 3: Gradient Filter by 45º degrees

The following filter takes the gradient of the image in off diagonal, i.e.,

$$h[n_1, n_2] = \begin{bmatrix} 0 & -1 & -1 \\ 1 & 0 & -1 \\ 1 & 1 & 0 \end{bmatrix}$$

Example 4: Lowpass Filter

The following filter is an approximation to the Gaussian lowpass filter, i.e.

$$h[n_1, n_2] = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{\sigma^2}}$$

Hence the filter is given as

$$h[n_1, n_2] = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

Lowpass filters are usually used to remove noise and smooth the image sequences.

Example 4: Laplacian, Highpass Filter

Highpass filters are used to find the edges in images. There are cases where one is only interested to find the edges in a specific direction such as x or y direction. On some other cases, edges are found independent of the direction. One example of a filter for this purpose is given below.

$$h[n_1, n_2] = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Example 5: Sobel Filter

This is a distinctly different filter form the previous LTI filters. This filter is nonlinear. While the LTI filters have some success in finding edges in images, one very successful filter for this purpose is the Sobel Filter. Sobel filter finds the gradient magnitude of images by using 3x3 filters. In many edge detection tasks, canny edge detector is used which implements the Sobel filter in addition to some other processes for suppression of noise artifacts. Sobel filtering involves a number of steps outlined below,

$$g_x[n_1, n_2] = x[n_1, n_2] * \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Above filter takes the gradient in x direction. The gradient in y direction is given below,

$$g_y[n_1, n_2] = x[n_1, n_2] * \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Then the gradient magnitude and angle of orientation are found as,

$$g[n_1, n_2] = \sqrt{g_x[n_1, n_2]^2 + g_y[n_1, n_2]^2}$$

$$\theta(n_1, n_2) = atan\left(\frac{g_y[n_1, n_2]}{g_x[n_1, n_2]}\right)$$

Above $g[n_1, n_2]$ is the gradient magnitude, $\theta(n_1, n_2)$ is the angle of orientation of the edge. Usually only the magnitude component is used in edge detection ignoring the phase, $\theta(n_1, n_2)$, term.

7.4 Linear-Phase 2D Filters

Phase is very important especially in image and video processing. Our eyes are more sensitive to phase changes than the magnitude changes. Therefore phase distortion is not desired in image processing during filtering. This is possible if we use linear-phase filters. 2D linear-phase filters satisfy the following relation,

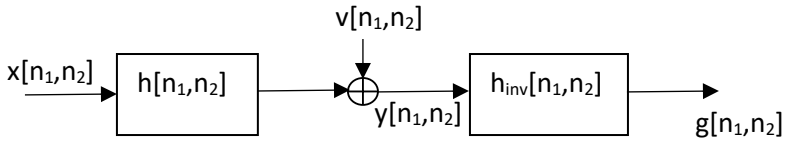
$$h[n_1, n_2] = h[N_1 - n_1 - 1, N_2 - n_2 - 1], \qquad 0 \le n_1 \le N_1 - 1, \qquad 0 \le n_2 \le N_2 - 1$$

Below two examples of 2D linear-phase filters are given as 3x3 and 4x4 filters respectively,

$$h[n_1, n_2] = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}, \quad h[n_1, n_2] = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 8 & 7 & 6 & 5 \\ 4 & 3 & 2 & 1 \end{bmatrix}$$

7.5 Inverse Filtering

Inverse filtering or deconvolution is used to obtain the original image when its filtered version is known. The following system structure shows the inverse filtering process.



In the above structure, $v[n_1,n_2]$ is the noise. The inverse filter design problem can be outlined as follows;

Problem: Given the output image, $y[n_1,n_2]$, and the filter $h[n_1,n_2]$, find the input image, $x[n_1,n_2]$.

The solution of the above problem is well known, namely the inverse filtering by $h[n_1,n_2]$. There is an extension of this problem, namely the blind deconvolution. In this case, only the output image is known and we need to find the input image. This is the reason why it is called as “blind”.

If we assume that noise $v[n_1,n_2]$ is zero and the filter $h[n_1,n_2]$ is minimum-phase, then the inverse filter in Fourier domain is given as,

$$H_{inv}[k_1,k_2] = \frac{1}{H[k_1,k_2]}$$

While the above result is straight forward, it is not applicable for all cases. One such case is when there is a zero on the unit circle. Consider the unit step function, u[n]. The DTFT of u[n] is given as,

$$U(e^{j\omega}) = \frac{1}{1-e^{-j\omega}} + \pi \sum_{k=-\infty}^{\infty} \delta[\omega + 2\pi k]$$

Hence if $H(z)=1-z^{-1}$, its inverse filter in Fourier domain is $H(e^{j\omega})=U(e^{j\omega})$. This observation does not solve the general problem. For example, if $X(e^{j\omega})$ has DC components, they can not be reconstructed. Furthermore if there is noise, noise amplification occurs.

Fortunately, there is a MSE optimum solution for the inverse filtering, namely the Wiener Inverse filter. Wiener inverse filter is given as,

$$H_{inv}[k_1,k_2] = \frac{|H[k_1,k_2]|^2}{|H[k_1,k_2]|^2 H[k_1,k_2] + \frac{1}{SNR[k_1,k_2]}}$$

In the above equation, $SNR[k_1,k_2]$ is the signal-to-noise ratio at the frequency index $[k_1,k_2]$. Usually a constant value for $SNR[k_1,k_2]$ is used for simplicity. However this results in a suboptimal solution. Nevertheless, Wiener inverse filter is usually used in image processing effectively.