Lectures	Lecture Topics
MODULE: 1	**Multirate Digital Signal Processing**
1.	Introduction
2.	Decimation by a factor D, Interpolation by a factor I
3.	Sampling rate conversion by rational factor I/D
4.	Filter Design and Implementation for sampling-rate
5.	Multistage implementation of sampling rate conversion
6.	Sampling rate conversion of Band pass signal
7.	Application of multi rate signal processing: design of phase shifters
8.	Implementation of narrowband lowpass filters
9.	Implementation of Digital filter banks.
10.	Filter Bank and Sub-band Filters and its applications.
MODULE: 2	**Linear prediction and Optimum Linear Filters**
11.	Innovations Representation of a stationary random process
12.	Forward Linear Prediction
13.	Backward Linear Prediction
14.	Solution of the normal equations
15.	Solution of the normal equations
16.	Properties of the linear prediction-error filters
17.	AR lattice- ladder filters
18.	ARMA lattice- ladder filters
19.	Wiener filter for filtering and Prediction: FIR Wiener Filter
20.	Orthogonality Principle in linear mean square estimation.
MODULE: 3	**Power Spectrum Estimation**
21.	Estimation of spectra from finite-duration observation of signals
22.	Non parametric method for power spectrum estimation
23.	Bartlett method
24.	Blackman method
25.	Turkey method
26.	parametric method for power estimation
27.	Yuke-Walker method

28.	Burg method
29.	MA model and ARMA model
MODULE: 4	**Adaptive Signal Processing**
30.	Introduction to adaptive signal processing
31.	Application of adaptive signal processing: System identification, Channel equalization, adaptive noise cancellation, adaptive line enhancer
32.	Application of adaptive signal processing: System identification, Channel equalization, adaptive noise cancellation, adaptive line enhancer
33.	Adaptive linear combiner
34.	Basics of Wiener filtering
35.	Widrow-Hopf Equation
36.	Least mean square algorithm
37.	Recursive least square algorithm
38.	variants of LMS algorithm: FX-LMS, Fast LMS, N-LMS, PN-LMS
39.	Continued... (FX-LMS, Fast LMS, N-LMS, PN-LMS)
40.	Design of Adaptive FIR & IIR filters

Text Book
1. Digital Signal Processing, Third Edition, J.G. Proakis and D.G. Manolakis, Prentice Hall.
2. Adaptive Signal Processing, B. Widrow and Stern,

Reference Book
1. Digital Signal Processing, by Sanjit K Mitra, new edition, TMH.
2. Digital Signal Processing, by Salivahanan, new edition, TMH.

Module 1.

Multirate Digital Signal Processing

INTRODUCTION

The process of converting a signal from a given rate to a different rate is called sampling rate conversion. Systems that employ multiple sampling rates in the processing of digital signals are called multirate digital signal processing systems. The two basic operations in a multirate system are decreasing (decimation) and increasing (interpolation) the sampling-rate of a signal.

DECIMATION

It is also called as down sampling. In this,the sampling rate of a discrete-time signal x(n) with sampling frequency Fs is reduced to a discrete-time signal y(n) of sampling frequency Fs/D, D is down sampling factor .The simplest way of doing so is to discard (D-1) samples for every D samples in original sequence.

Mathematically:

If x(n) is the original discrete-time signal

Then $y(n) = x(nD)$

So, in z domain,

$Y(z) = \sum_{n=-\infty}^{\infty} x(nD)\, z^{-n}$

Let $nD = p \Rightarrow n = p/D$

So $Y(z) = \sum_{n=-\infty}^{\infty} x(p) (z^{1/D})^{p}$

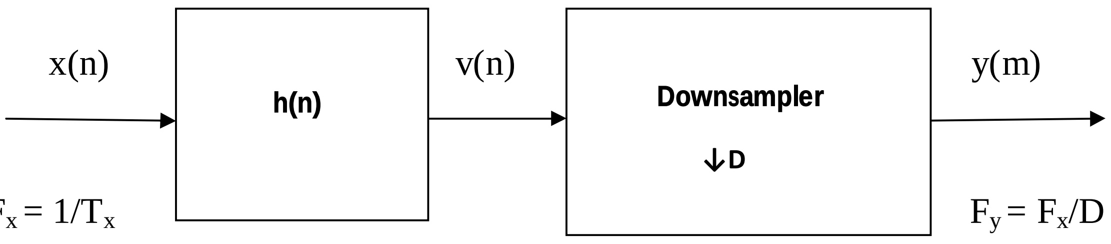
$\Rightarrow Y(z) = X(z^{1/D})$

On unit circle, $Y(\omega) = X(\omega/D)$

It seems hence that spectrum of original signal gets stretched as an effect of down sampling. Hence, an anti-aliasing digital filter is used before down-sampling to prevent aliasing.

The input sequence *x(n)* is passed through a lowpass filter, characterized by the impulse response *h(n)* and a frequency response *$H_D(\omega)$,* which ideally satisfies the condition

$$H_D(\omega) = \begin{cases} 1, & |\omega| \leq \pi/D \\ 0, & \text{otherwise} \end{cases}$$



Decimation by a factor D

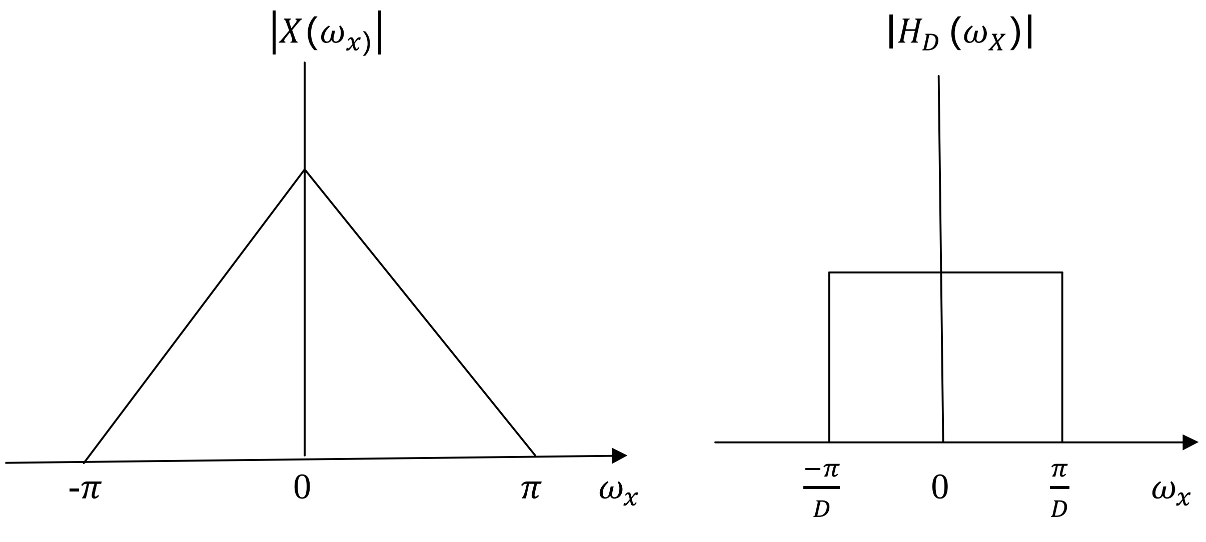
Time domain expression:

v(n) = x(n) * h(n)

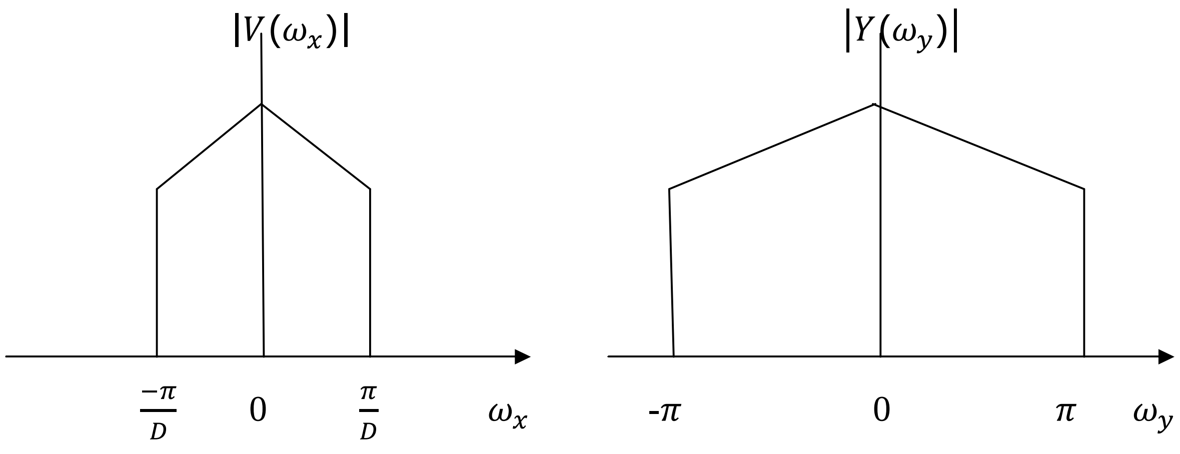
$= \sum_{n=-\infty}^{\infty} h(k)x(n-k)$

y(m) = v(mD)

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



Spectra of signals during decimation by factor



Spectra of signals during decimation by factor D

INTERPOLATION

It is also called as up sampling. In this,the sampling rate of a discrete-time signal x(n) with sampling frequency Fs is increased to a discrete-time signal y(n) of sampling frequency IFs, I is up sampling factor .The simplest way of doing so is to insert (I - 1) samples for every I samples in original sequence.

Mathematically:

If x(n) is the original discrete-time signal

Then y(n) = x(n/I)

So, in z domain,

$Y(z) = \sum_{n=-\infty}^{\infty} x(n/I)\, z^{-n}$

Let n/I= p => n= pI

So $Y(z) = \sum_{n=-\infty}^{\infty} x(p)(z^{I})^{p}$

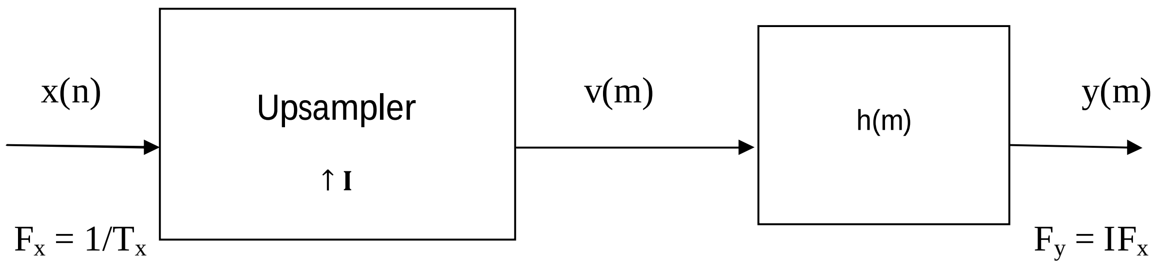
$\Rightarrow Y(z) = X(z^{I})$

On unit circle, Y(ω) = X(ωI)

So interpolation causes the original spectrum to get compressed by I-folds. It yields undesirable replicas in the signal's frequency spectrum. Hence, it is necessary to remove these replicas from the frequency spectrum. So the expansion process is followed by a unique digital low-pass filter called an anti-imaging filter.

$$H_I(\omega) = \begin{cases} C & ,0 \le |\omega| \le \pi/I \\ 0 & ,\text{otherwise} \end{cases}$$

$$Y(\omega) = \begin{cases} CX(\omega I) & ,0 \le |\omega| \le \pi/I \\ 0 & ,\text{otherwise} \end{cases}$$



Interpolation of signal by a factor I

Time domain expression:

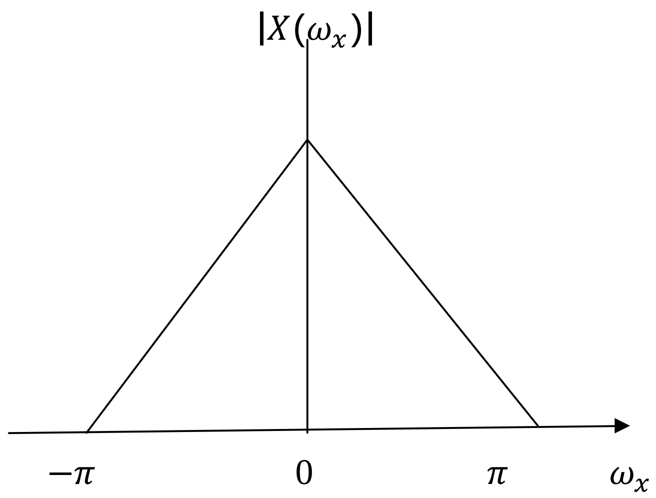
$$v(m) = \begin{cases} x\left(\frac{m}{I}\right) & , m = 0, \pm I, \pm 2I, \ldots. \\ 0 & , otherwise \end{cases}$$

*y(m) = v(m) * h(m)*

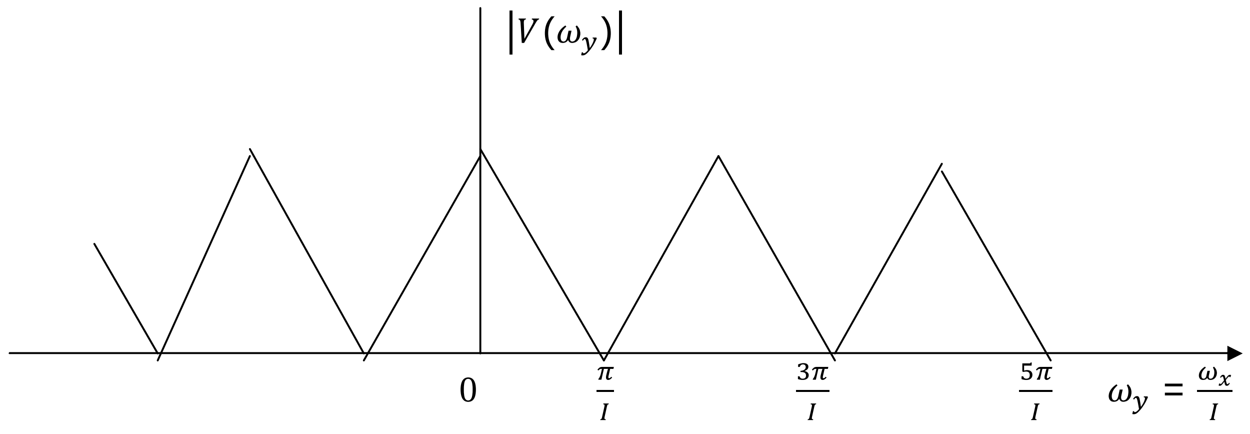
$$= \sum_{m=-\infty}^{\infty} h(m-k)v(k)$$

Since *v(k)=0* except at multiples of *I*, where *v(kI)=x(k)* , therefore

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



Spectra of x(n)

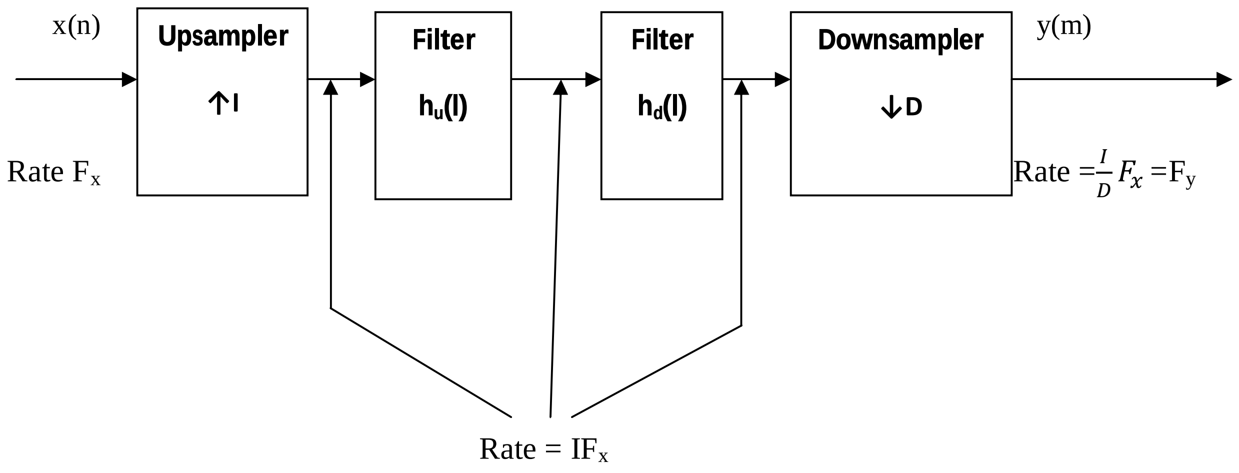


Spectra of v(n)

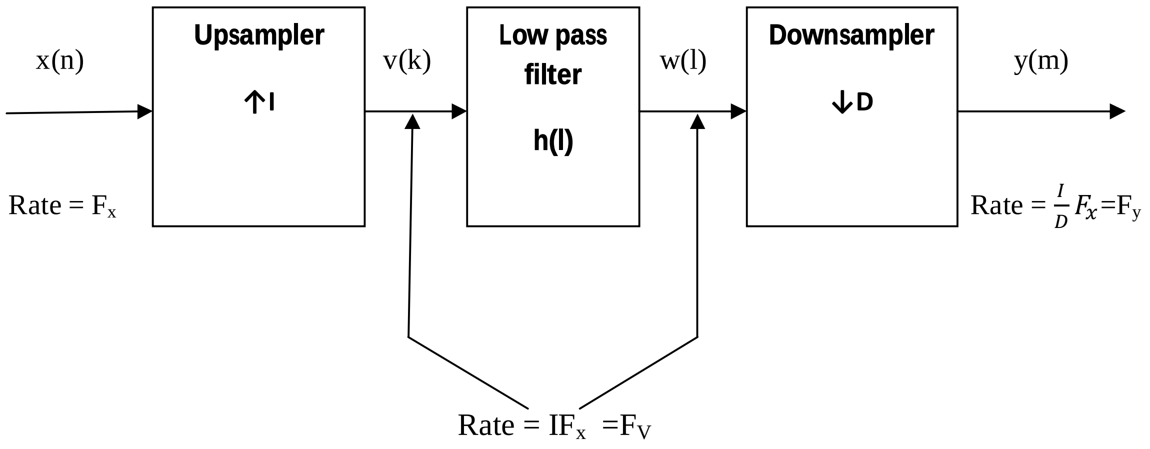
Sampling Rate Conversion by a Rational Factor I/D

We can achieve sampling rate conversion by a rational factor I/D by first performing interpolation by a factor I and then decimating the output of the interpolator by the factor D. The importance of performing interpolation first and decimation second , is to preserve the desired spectral characteristics of x(n). Since the two filters are operated at the same sampling rate IF_x, the two filters can be replaced by a single lowpass filter with impulse response *h(l)* .The frequency response $H(\omega_v)$ of the combined filter is given as

$$H(\omega_v) = \begin{cases} I & ,0 \le |\omega_v| \le \min\left(\frac{\pi}{D}, \frac{\pi}{I}\right) \\ 0 & , Otherwise \end{cases}$$



Method of sampling rate conversion by factor I/D



Method of sampling rate conversion by factor I/D

Where $\omega_v = 2\pi F/F_v = 2\pi F/IF_x = \omega_x / I$

$$v(l) = \begin{cases} x\left(\frac{l}{I}\right) & , l = 0, \pm I, \pm 2I, \ldots \\ 0 & , otherwise \end{cases}$$

$$w(l) = \sum_{k=-\infty}^{\infty} h(l-k)v(k)$$

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

$$y(m) = w(mD)$$

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

Filter Design and Implementation for Sampling-rate Conversion

It includes various structures like:

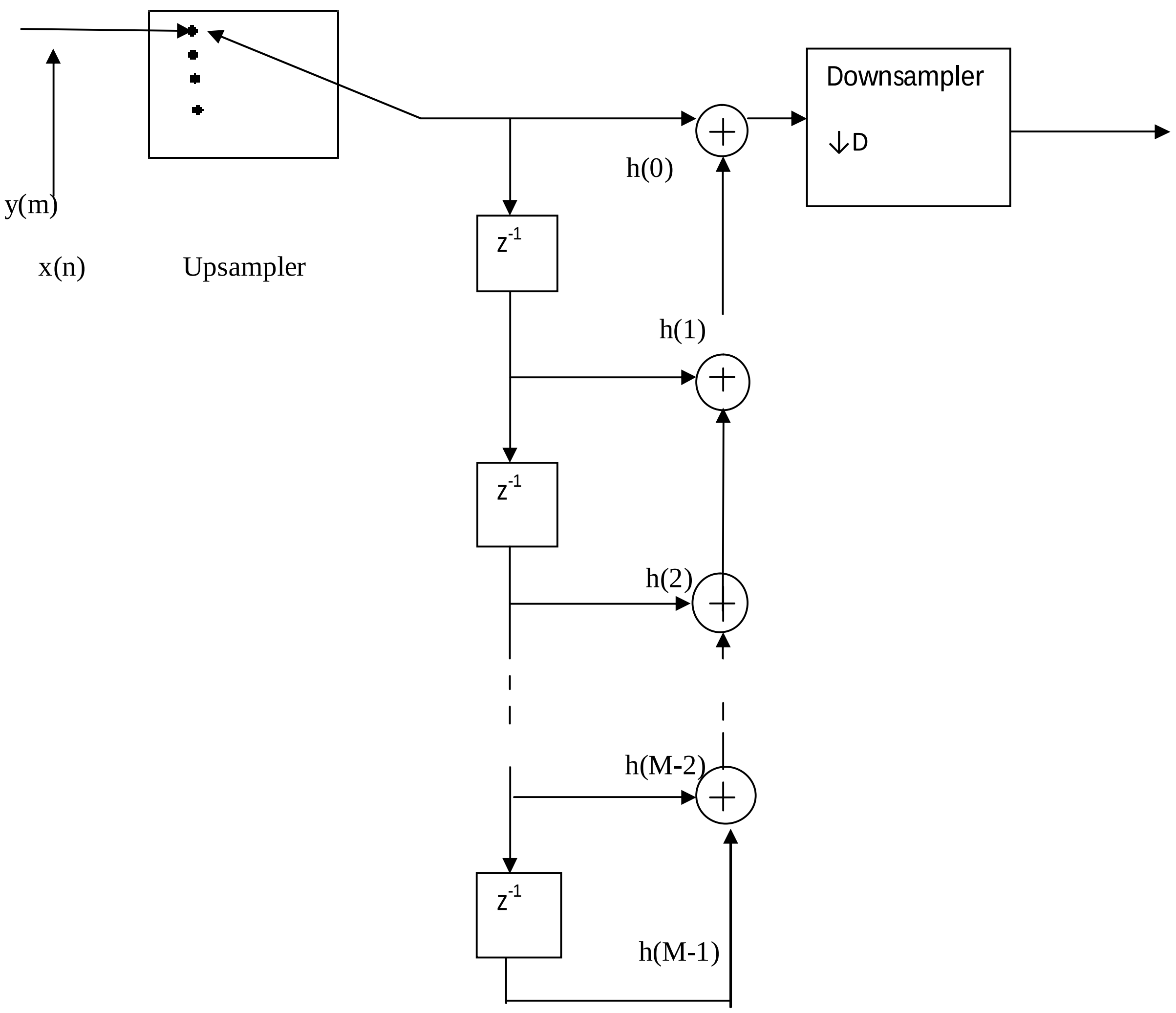
- Direct form FIR filter
- Polyphase filter
- Time-variant filter

Direct Form FIR Filter Structure

This is the simplest realization with system function

$H(z) = \sum_{k=0}^{M-1} h(k) z^{-k}$

where h(k) is the unit sample response of the FIR filter.



Although the direct form FIR realization is simple ,it is also very inefficient. The inefficiency results from the fact that the upsampling introduces I-1 samples between successive points of the input signal. If I is large, most of the signal components in the FIR filter are zero. Consequently, most of the multiplications and additions result in zeros. Furthermore, the downsampling process at the output of the filter implies that only one out of every D output samples is required at the output of the filter.

Polyphase Filter Structure

The polyphase structure is based on the fact that any system function can be split as

$H(z) = \ldots\ldots + h(0) \qquad + h(M) z^{-M} + \ldots\ldots\ldots\ldots$

$\ldots\ldots + h(1)z^{-1} \qquad + h(M+1)\,z^{-(M+1)} + \ldots\ldots$

.

.

.

$\ldots\ldots + h(M-1)\,z^{-(M-1)} + h(2M-1)\,z^{-(2M-1)} + \ldots\ldots\ldots$

If we next factor out the term $z^{-(i-1)}$ at the ith row, we obtain

$H(z) = [\ldots\ldots + h(0) \qquad + h(M)\,z^{-M} + \ldots\ldots]$

$\quad + z^{-1}[\ldots.. + h(1) \qquad + h(M+1)\,z^{-M} + \ldots\ldots.]$

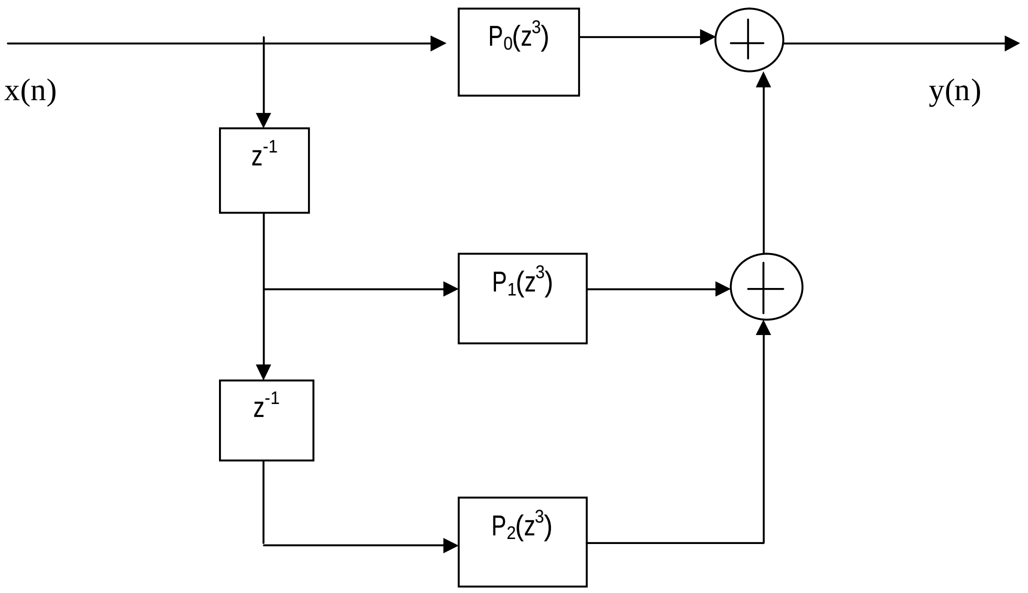
.

.

$\quad + z^{-(M-1)}[\ldots.. + h(M-1) + h(2M-1)\,z^{-M} + \ldots\ldots\ldots..]$

This implies $\boxed{H(z) = \sum_{i=0}^{M-1} z^{i}\, P_i(z^M)}$

where $P_i(z) = \sum_{n=-\infty}^{\infty} h(nM+i)z^{-n}$



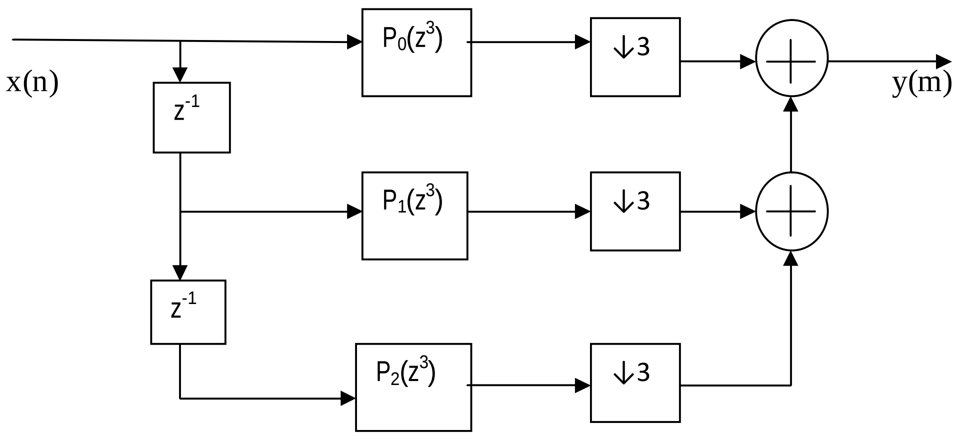
Block diagram of polyphase filter structure for M = 3

The output sequence for M = 3 will be:

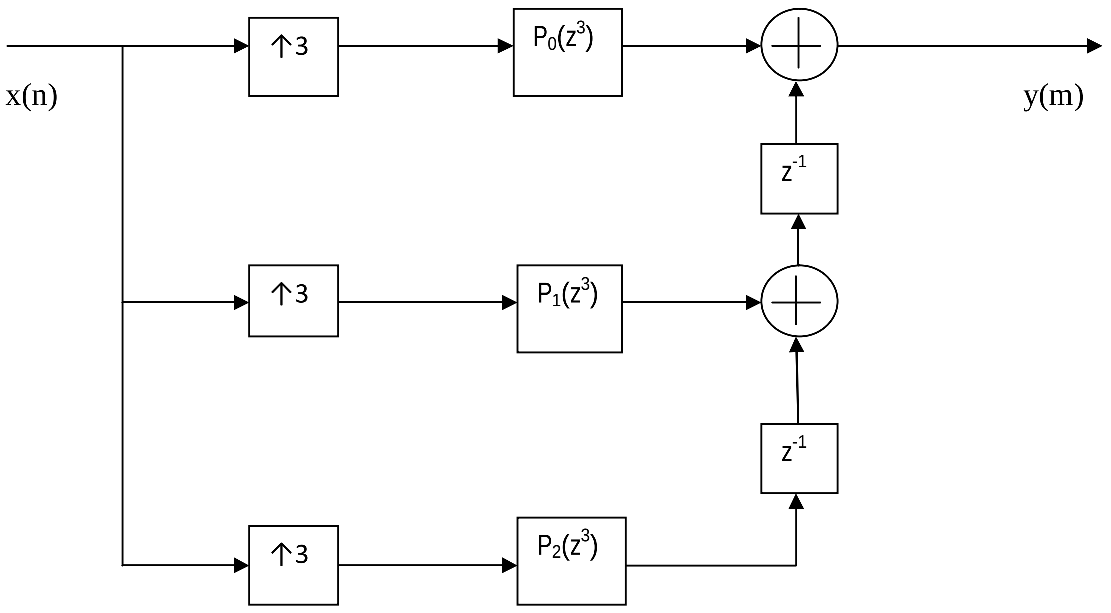
$$Y(z) = H(z)X(z)$$

$$= P_0(z^3)\,X(z) + z^{-1}\,P_1(z^3)\,X(z) + z^{-2}\,P_2(z^3)\,X(z)$$

Implementation of a decimation system using a polyphase structure



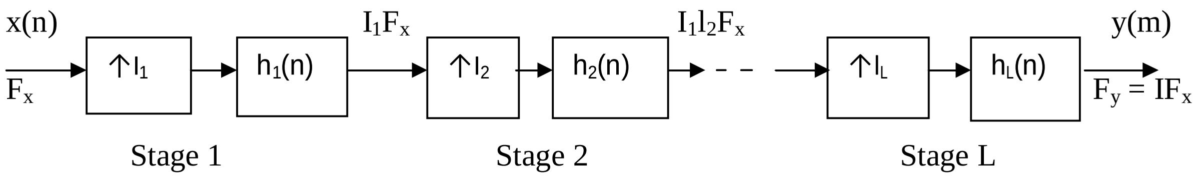
Implementation of a interpolation system using a polyphase structure



Multistage Implementation of Sampling-rate Conversion

Let us consider interpolation by a factor I>>1 and let us assume that I can be factored into a product of positive integers as

$$I = \prod_{i=1}^{L} I_i$$



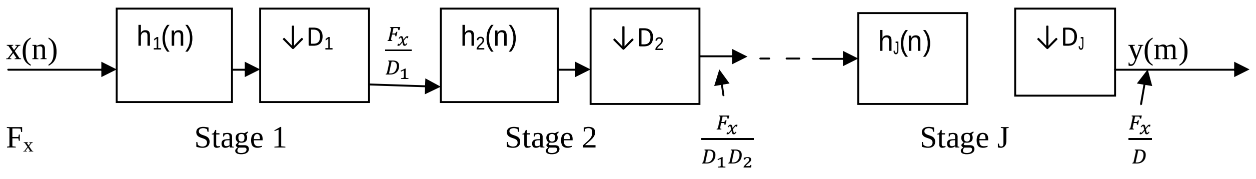
Multistage implementation of interpolation by a factor I

Interpolation by a factor I can be accomplished by cascading L stages of interpolation and filtering. The filter in each of the interpolators eliminates the images introduced by the upsampling process in the corresponding interpolator.

In a similar manner, decimation by a factor D , where D may be factored into a product of positive integers as

$$D = \prod_{i=1}^{J} D_i$$

can be implemented as a cascade of J stages of filtering and decimation.



Multistage implementation of decimation by a factor D

Sampling Rate Conversion of Bandpass Signals

Any band pass signal has an equivalent low pass representation , obtained by a simple frequency translation of the band pass signal.

An analog band pass signal can be represented as

$x(t) = A(t)\cos[2\pi F_c t + \theta(t)]$

$= A(t)\cos\theta(t)\cos(2\pi F_c t) - A(t)\sin\theta(t)\sin(2\pi F_c t)$

$= u_c(t)\cos(2\pi F_c t) - u_s(t)\sin(2\pi F_c t)$

$= \mathrm{Re}[x_l(t)\, e^{-j2\pi F_c t}]$

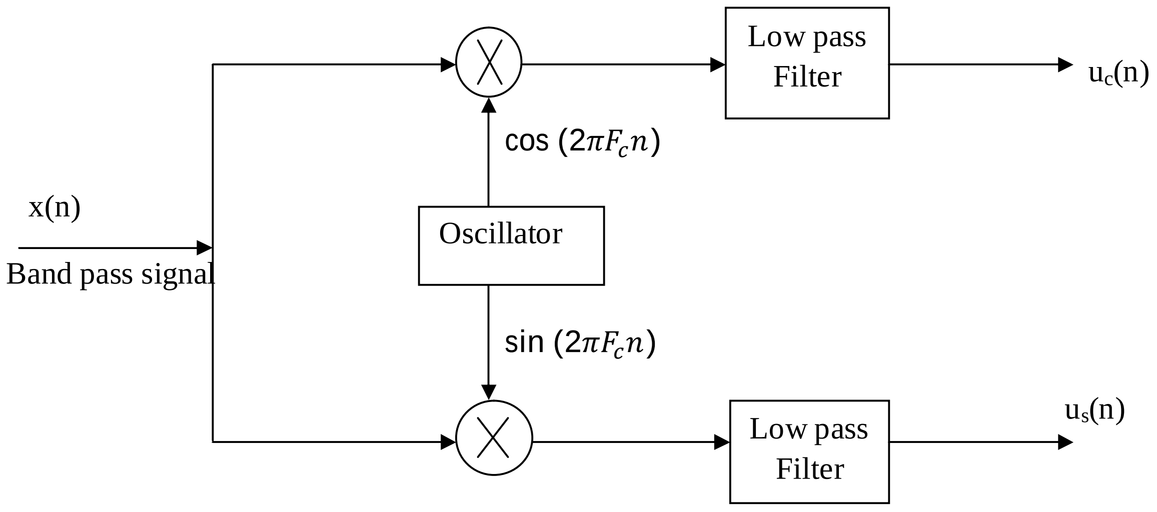
where $u_c(t) = A(t)\cos\theta(t)$

$u_s(t) = A(t)\sin\theta(t)$

$x_l(t) = u_c(t) + j\, u_s(t)$

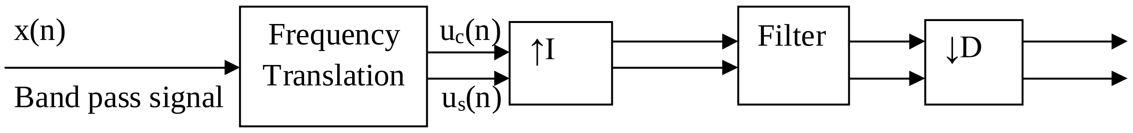
A(t) is called the amplitude or envelope of the signal , $\theta(t)$ is the phase , $u_c(t)$ and $u_s(t)$ are called the quadrature components of the signal.

Physically the translation of x(t) to low pass involves multiplying x(t) by the quadrature carriers $\cos(2\pi F_c t)$ and $\sin(2\pi F_c t)$ and then low pass filtering the two products to eliminate the frequency components generated around the frequency $2F_c$.



(Conversion of a band pass signal to low pass)

The mathematical equivalence between the band pass signal and its low pass representation provides one method for altering the sampling rate of the signal .



Sampling rate conversion of a band pass signal

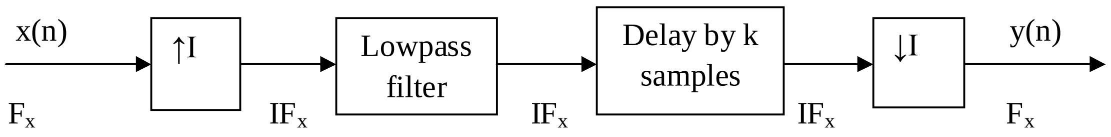
The filter in the above diagram has the frequency response characteristics :

$$H(\omega) = \begin{cases} I & ,0 \le |\omega| \le \min\left(\frac{\omega_B}{2D}, \frac{\omega_B}{2I}\right) \\ 0 & , otherwise \end{cases}$$

where ω_B is the bandwidth of the discrete – time band pass signal ($\omega_B \le \pi$) .

Applications of Multirate Signal Processing

1. **Design of Phase Shifters:**
   A delay of $(k/I)T_x$ can be achieved by sample rate conversion method without introducing any significant distortion in the signal. The sampling rate is increased by a factor I using standard interpolator. The lowpass filter eliminates the images in the spectrum of the interpolated signal , and its output is delayed by k samples at the sampling rate IF_x . The delayed signal is decimated by a factor D = I. Thus we have achieved the desired delay of $(k/I)T_x$.



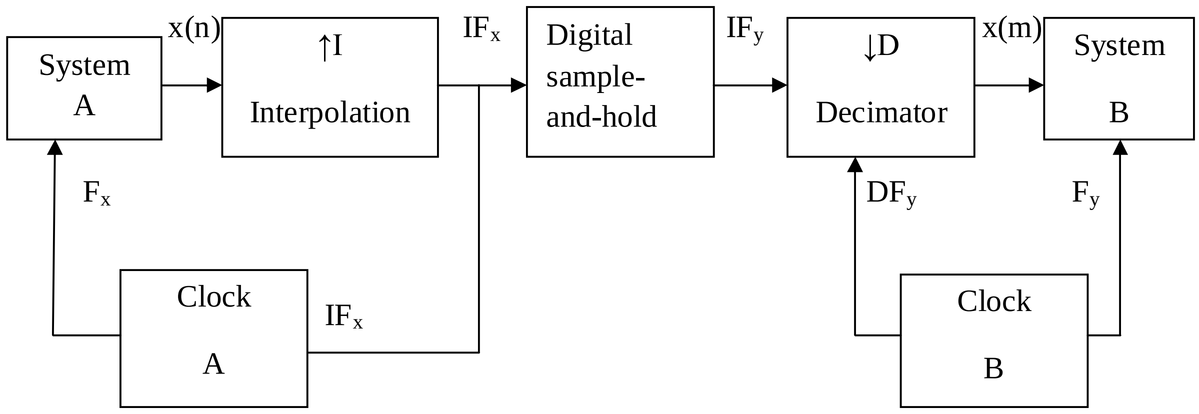
x(n) upsampled= x(n/I)
x(n/I) delayed by k samples= x((n-k)/I)
x((n-k)/I) downsampled = x(n-(k/I))

If $F[x(n)] = X(\omega)$,Then $F[x(n-k)] = e^{-j\omega k}X(\omega)$

So $F[x(n-k/I)] = e^{-j\omega k/I}X(\omega)$

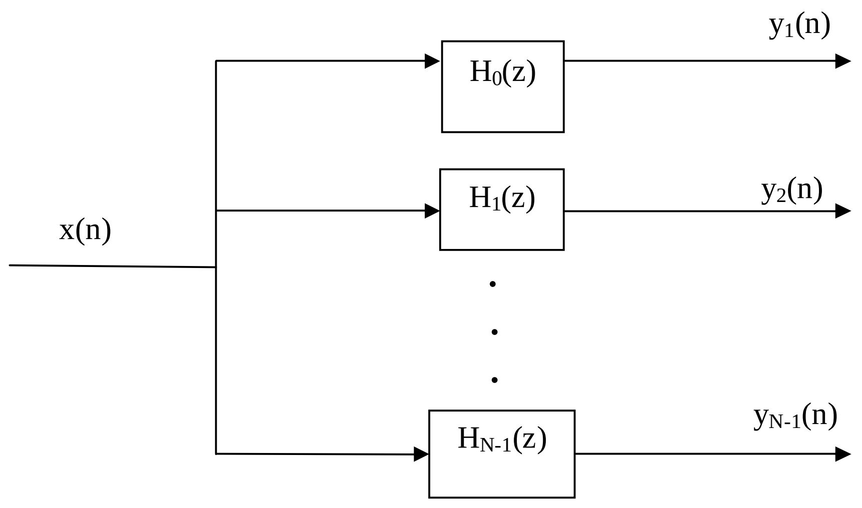
Hence the original spectrum of X(ω) gets a phase shift of $\phi = \frac{-\omega k}{I}$

2. **Interfacing of Digital Systems with Different Sampling Rates :**
   Let us consider interfacing the two systems with independent clocks. The output of the system A at rate F_x is fed to an interpolator which increases the sampling rate by I. The output of the interpolator is fed at the rate lF_x to a digital sample-and-hold which serves as the interface to system B at the high sampling rate IF_x . Signals from the digital sample-and-hold are read out into system B at the clock rate DF_y of system B . Thus the output rate from the sample-and-hold is not synchronized with the input rate.

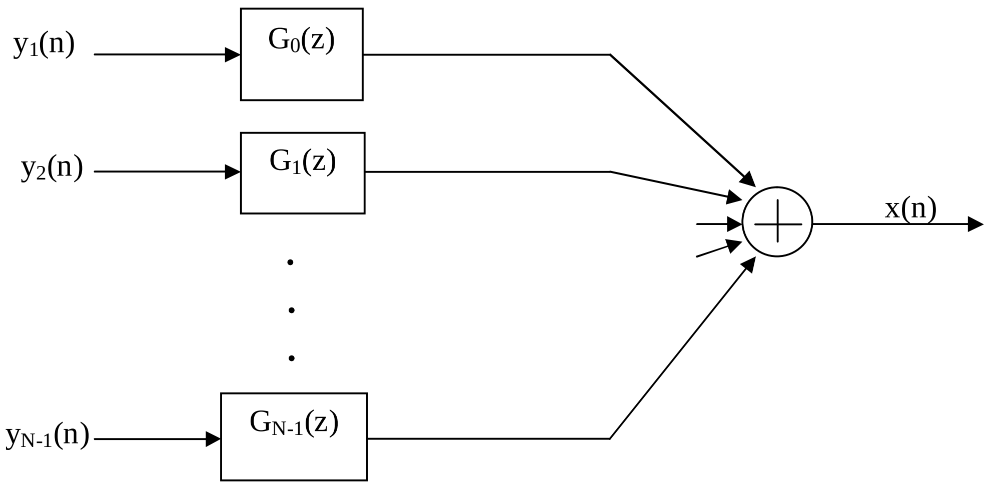


3. **Implementation of Narrowband Lowpass Filters:**
   A lowpass , linear-phase FIR filter may be more efficiently implemented in a multistage decimator-interpolator configuration. To be more specific , we can employ a multistage implementation of a decimator of size D , followed by a multistage implementation of an interpolator of size I , where I = D.

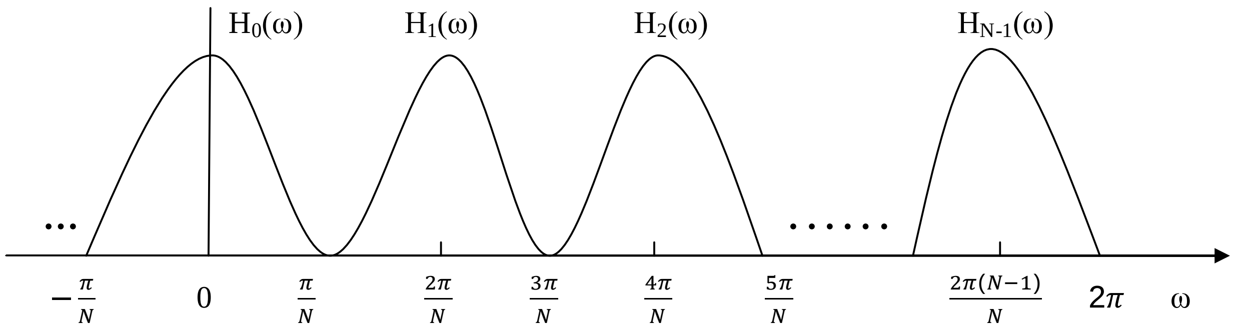
4. **Implementation of Digital Filter Banks:**
   Filter banks are generally categorized as two types , analysis filter banks and synthesis filter banks . An analysis filter bank consists of a set of filters , with system function $\{H_k(z)\}$, arranged in parallel. The frequency response characteristics of this filter bank split the signal into a corresponding number of subbands. On the other hand, a synthesis filter consists of a set of filters with system function $\{G_k(z)\}$, with corresponding inputs $\{y_k(n)\}$. The outputs of filters are summed to form the synthesized signal $\{x(n)\}$.



(Analysis filter bank)



(Synthesis filter bank)



(Frequency response characteristics of N filters)

5. **Subband Filters :**
Subband coding is a method where the signal is subdivided into several frequency bands and each band is digitally encoded separately.

Let us assume a signal with sampling rate F_s . The first frequency subdivision splits the signal spectrum into two equal-width segments , a lowpass signal ($0 \leq F \leq F_s/4$) and a highpass signal ($Fs/4 \leq F \leq Fs/2$). . The second frequency subdivision splits the lowpass signal from the first stage into two equal bands , a lowpass signal ($0 \leq F \leq Fs/8$) and a highpass signal ($Fs/8 \leq F \leq Fs/4$). Finally, the third subdivision splits the lowpass signal from the second stage into two equal-bandwidth signals. Thus the signal is subdivided into four frequency bands , covering three octaves.

Module 2.

Linear Prediction and Optimum Linear Filters

Innovations Representation of a stationary random process:

A wide-sense stationary random process can be represented as the output of a causal and casually invertible linear system excited by a white noise process. Let the wide-sense stationary random process be {x(n)} with autocorrelation function $\gamma_{xx}(m)$, and power spectral density $\Gamma_{xx}(f)$, $|f| < \frac{1}{2}$. Then it can be shown that :

$$\Gamma_{xx}(z) = \sigma_w{}^2 H(z)H(z^{-1})$$

Or on the unit circle,

$$\Gamma_{xx}(f) = \sigma_w{}^2 |H(f)|^2$$

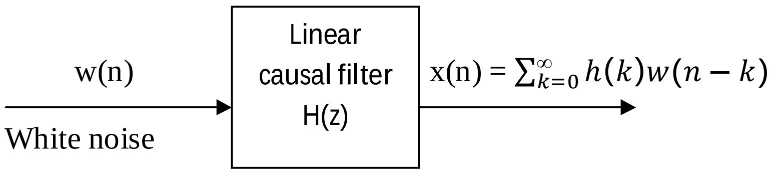
H(f) hence represents a filter which when excited by white noise *w(n)* of power spectral density $\sigma_w{}^2$, gives an output *{x(n)}* with power spectral density $\sigma_w{}^2|H(f)|^2$. The random process x(n) is generated by passing white noise input sequence w(n) through a linear causal filter H(z) :

$$\mathrm{H(z)} = \frac{B(z)}{A(z)} = \frac{\sum_{k=0}^{q} b_k z^{-k}}{1 + \sum_{k=1}^{p} a_k z^{-k}} \qquad |z| > r_1 < 1$$

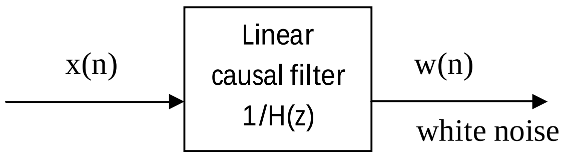
where the polynomials B(z) and A(z) have roots that fall inside the unit circle in the z-plane. $\{b_k\}$ and $\{a_k\}$ are filter coefficients that determine the location of the zeros and poles of H(z) , respectively. Thus the output *x(n)* is related to the input *w(n)* by the difference equation

$$\mathrm{x(n)} + \sum_{k=1}^{p} a_k x(n-k) = \sum_{k=0}^{q} b_k w(n-k)$$

The representation of the stationary random process be *{x(n)}*, as the output of an IIR filter with transfer function H(z) and excited by a white noise process {w(n)} is called as **Wold representation**. The stationary random process *{x(n)}* can be transformed into white noise process by passing {x(n)} through a linear filter with system function 1/H(z) . This filter is called a ***noise whitening filter***. Its output, denoted as {w(n)} is called the ***innovations process*** associated with the stationary random process {x(n)}.


w(n)
White noise
Linear causal filter H(z)
x(n) = $\sum_{k=0}^{\infty} h(k)w(n-k)$

Filter for generating the random process x(n) from white noise



Noise Whitening Filter

Rational Power Spectra

When the power spectral density of a stationary random process is a rational function given by:

$$\Gamma_{xx}(z) = \sigma_w^2 \frac{B(z)B(z^{-1})}{A(z)A(z^{-1})}$$

and B(z) and A(z) have roots that fall inside the unit circle, then the filter H(z) for generation of {x(n)} is also rational expressed as :

$$H(z) = \frac{B(z)}{A(z)} = \frac{\sum_{k=0}^{q} b_k z^{-k}}{1+\sum_{k=1}^{p} a_k z^{-k}} \qquad |z| > r_1 < 1$$

The H(z) is causal, stable and minimum phase linear filter. The noise whitening filter 1/H(z) is also causal, stable and minimum phase linear filter. We can have three special cases on the basis of coesfficients a_k*'s* and b_k*'s*.

Autoregressive (AR) process:

$b_0 = 1$, $b_k = 0$, $k > 0$. In this case , the linear filter $H(z) = 1/A(z)$ is an all-pole filter and the difference equation for the input-output relationship is

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

The noise-whitening filter for generating the innovations process is an all-zero filter.

Moving average (MA) process:

$a_k = 0$, $k \geq 1$. In this case , the linear filter $H(z) = B(z)$ is an all-zero filter and the difference equation for the input-output relationship is

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

The noise-whitening filter for the MA process is an all-pole filter.

Autoregressive, moving average (ARMA) process:

In this case , the linear filter H(z) = B(z)/A(z) has both finite poles and zeros in the z-plane and the corresponding difference equation for the input-output relationship is

$$x(n) + \sum_{k=1}^{p} a_k x(n-k) = \sum_{k=0}^{q} b_k w(n-k)$$

The inverse system for generating the innovations process from x(n) is a pole-zero system of the form 1/H(z) = A(z)/B(z).

Forward and Backward Linear Prediction :

Forward Linear Prediction:

In forward linear prediction , a future value of stationary random process is predicted from observation of past values of the process . The one-step forward linear predictor forms the prediction of the value x(n) by weighted linear combination of the past values x(n-1) , x(n-2) , ... , x(n-p) . Hence the linearly predicted value of x(n) is

$$\hat{x}(n) = -\sum_{k=1}^{p} a_p(k)\, x(n-k)$$

where the $\{-a_p(k)\}$ represent the weights in the linear combination . These weights are called the prediction coefficients of the one-step forward linear predictor of order p .

The difference between the value x(n) and the predicted value x(n) is called the ***forward prediction error*** , denoted as $f_p(n)$:

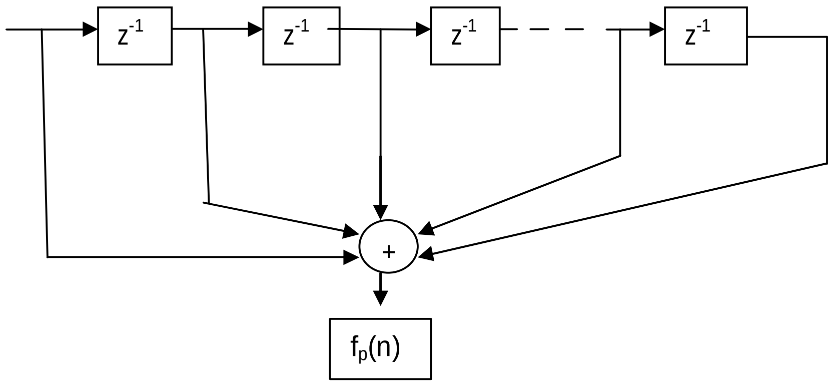
$$f_p(n) = x(n) - \hat{x}(n)$$

$$= x(n) + \sum_{k=1}^{p} a_p(k)\, x(n-k)$$

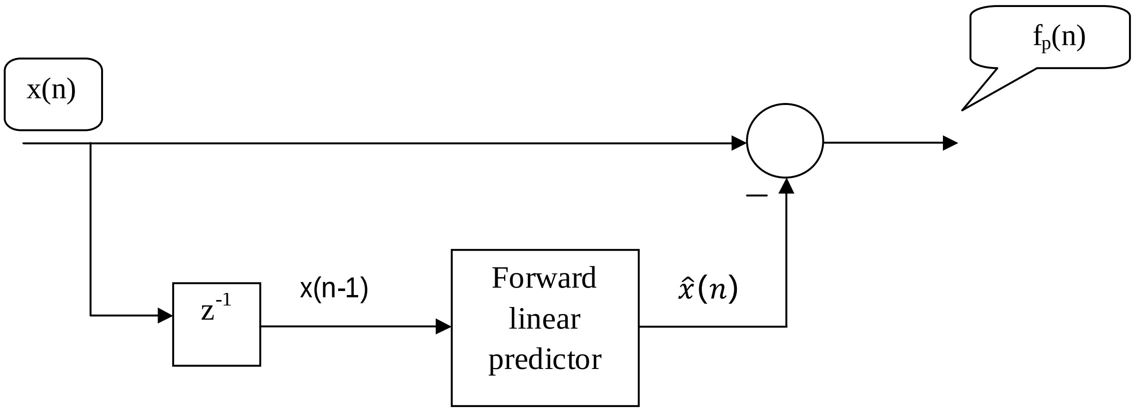
The direct form FIR filter realization to find the forward prediction error can hence be written as:

$$A_p(z) = \sum_{k=0}^{p} a_p(k) z^{-k}$$

where by definition $a_p(0)=1$. The equivalent direct for FIR filter realization structure is drawn below :



Direct Form Structure Of Prediction Error Filter



Forward linear prediction

Backward Linear Prediction

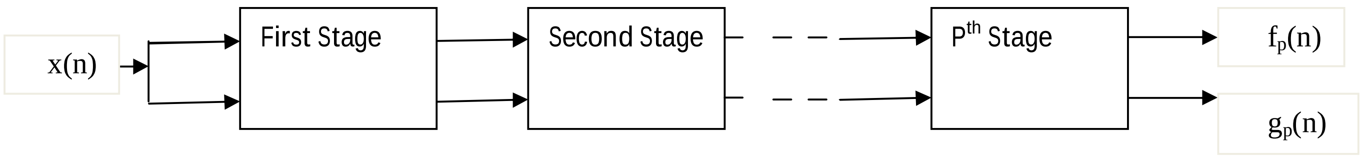
Suppose we have the data sequence x(n) , x(n-1) , ... , x(n - p + 1) from a stationary random process and we wish to predict the value x(n-p) of the process . In this case a one-step backward linear predictor of order p is employed. Hence

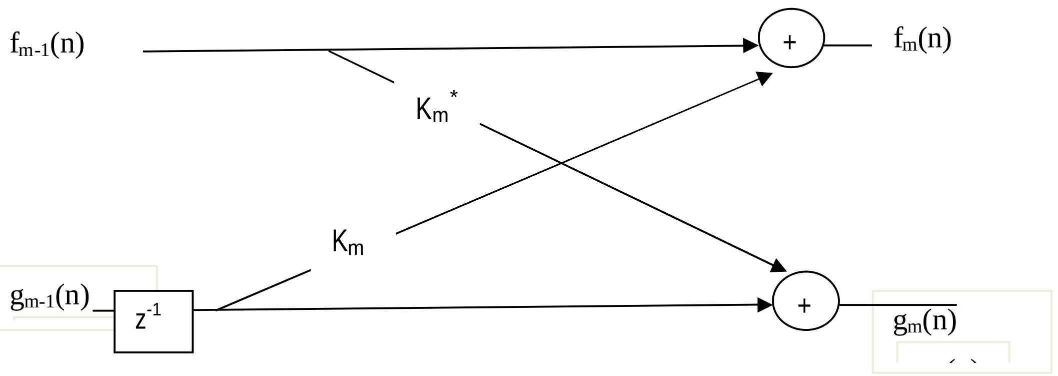
$$\hat{x}(n-p) \;=\; -\sum_{k=0}^{p-1} b_p\,(k)\;x(n-k)$$

The difference between the value x(n-p) and the estimate $\hat{x}(n-p)$ is called ***the backward prediction error*** , denoted as $g_p(n)$:

$$g_p(n) = x(n - p) + \sum_{k=0}^{p-1} b_p\,(k)\;x(n-k)$$

$$= \sum_{k=0}^{p} b_p\,(k)\;x(n-k)\;, \qquad b_p(p) = 1$$





p-stage Lattice filter for forward and backward prediction

Normal Equations :

The forward prediction error is given as

$$f_p(n) = \sum_{k=0}^{p} a_p\,(k)\; x(n-k)$$

The corresponding z-transform relationship is

$$F_p(z) = A_p(z)\, X(z)$$

$$\Rightarrow A_p(z) = \frac{\mathrm{Fp(z)}}{X(z)} = \frac{\mathrm{Fp(z)}}{\mathrm{F0(z)}}$$

The mean square value of forward linear prediction error $f_p(n)$ is

$$\varepsilon_p^f = E[|f_p(n)|^2]$$

$$= \gamma_{xx}(0) + 2\Re[\sum_{k=1}^{p} a_p^*(k)\,\gamma_{xx}(\mathrm{k})] + \sum_{k=1}^{p}\sum_{l=1}^{p} a_p^*(l) a_p(k)\,\gamma_{xx}(\mathrm{l-k})$$

ε_p^f is a quadratic function of the predictor coefficients and its minimum leads to the set of linear equations

$$\boxed{\gamma_{xx}(l) = -\sum_{k=1}^{p} a_p(k)\,\gamma_{xx}(l-k)} \quad , \quad l=1, 2, \ldots, p$$

These are called the normal equations for the coefficients of the linear predictor.

The minimum mean-square prediction error is simply

$$\min[\varepsilon_p^f] \equiv E_p^f = \gamma_{xx}(0) + \sum_{k=1}^{p} a_p(k)\,\gamma_{xx}(l-k)$$

Solutions of the Normal Equations:

The normal equations may be expressed in the compact form

$$\sum_{k=0}^{p} a_p(k)\,\gamma_{xx}(l-k) = 0 \qquad l = 1, 2, \ldots, p$$

$$a_p(0) = 1$$

If we augment the minimum MSE expression with the above equation , we get

$$\sum_{k=0}^{p} a_p(k)\,\gamma_{xx}(l-k) = \begin{cases} E_p^f & , l = 0 \\ 0 & , l = 1, 2, \ldots, p \end{cases}$$

The Levinson-Durbin Algorithm:

This algorithm exploits the symmetry in the autocorrelation matrix

$$\Gamma_p = \begin{bmatrix} \gamma_{xx}(0) & \gamma_{xx}^*(1) & \cdots & \gamma_{xx}^*(p-1) \\ \gamma_{xx}(1) & \gamma_{xx}(0) & \cdots & \gamma_{xx}^*(p-2) \\ \vdots & \vdots & \cdots & \vdots \\ \gamma_{xx}(p-1) & \gamma_{xx}(p-2) & \cdots & \gamma_{xx}(0) \end{bmatrix}$$

The solution to the first-order predictor is

$$a_1(1) = -\frac{\gamma_{xx}(1)}{\gamma_{xx}(0)}$$

The next step is to solve for the coefficients $\{a_2(1), a_2(2)\}$ of the second order predictor. The two equations obtained from the normal equation are:

$$a_2(1)\gamma_{xx}(0) + a_2(2)\,\gamma_{xx}^*(1) = -\gamma_{xx}(1)$$

$$a_2(1)\gamma_{xx}(1) + a_2(2)\gamma_{xx}(0) = -\gamma_{xx}(2)$$

By using above equation and the expression for $a_1(1)$, we obtain the solution

$$a_2(2) = -\frac{\gamma_{xx}(2) + a_1(1)\gamma_{xx}(1)}{\gamma_{xx}(0)[1-|a_1(1)|^2]}$$

Properties of the linear prediction-error filters :

1. **Minimum-phase property of the forward prediction-error filter:**
   If a random process consists of a mixture of a continuous power spectral density and a discrete spectrum , the prediction-error filter must have all its roots inside the unit circle.

2. **Maximum-phase property of the backward prediction-error filter:**
   The system function for the backward prediction-error filter of order p is
   $$B_p(z) = z^{-p} A_p^*(z^{-1})$$
   where $A_p(z)$ is the system function for the forward prediction filter.
   Consequently, the roots of $B_p(z)$ are the reciprocals of the roots of the forward prediction-error filter with system function $A_p(z)$. Hence if $A_p(z)$ is minimum phase , then $B_p(z)$ is maximum phase.

3. **Orthogonality of the backward prediction errors:**
   The backward prediction errors $\{g_m(k)\}$ from different stages in the FIR lattice filter are orthogonal. That is ,
   $$E[g_m(n)\, g_l^*(n)] = \begin{cases} 0 & ,0 \le l \le m-1 \\ E_m^b & ,l = m \end{cases}$$

4. **Whitening property:**
   The prediction-error filter attempts to remove the correlation among the signal samples of the input process. The response of the prediction-error filter is a white noise sequence . Thus it whitens the input random process and is called a whitening filter.

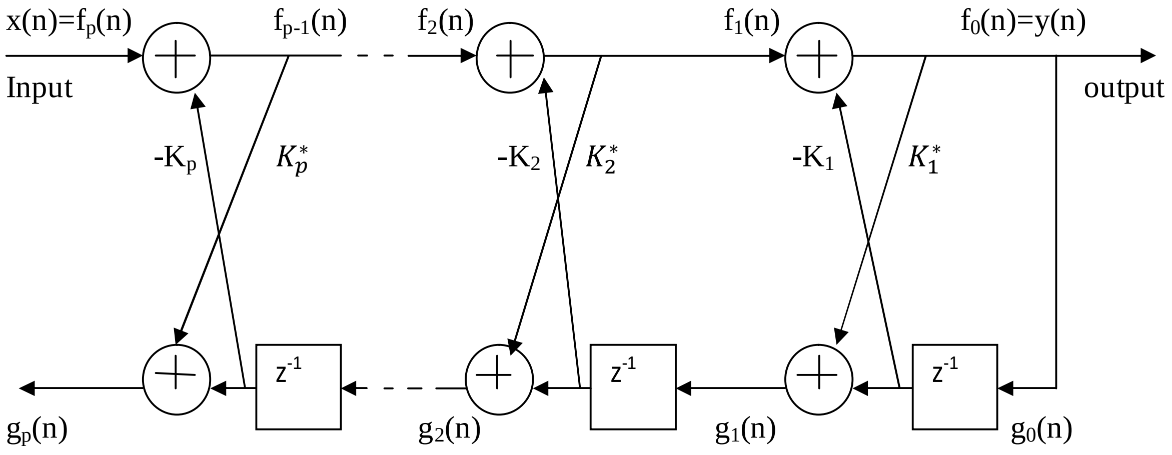
AR Lattice Structure:

The difference equation for AR process is given by:

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

If $f_p(n)$ is the forward prediction error , $g_p(n)$ is the backward prediction error and $\{K_m\}$ are the reflection coefficients then we have:

$$x(n) = f_p(n)$$
$$f_{m-1}(n) = f_m(n) - K_m\, g_{m-1}(n-1)\,, \quad m = p, p-1, \ldots, 1$$
$$g_m(n) = K_m^*\, f_{m-1}(n) + g_{m-1}(n-1)$$
$$y(n) = f_0(n) = g_0(n)$$



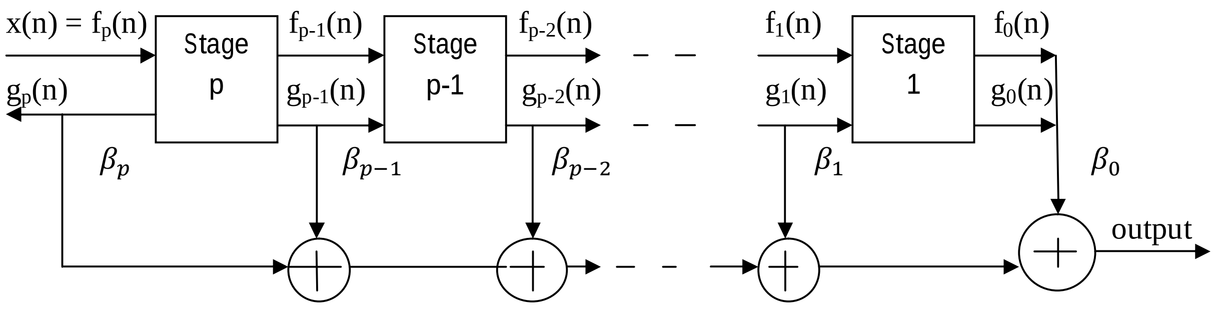
Lattice structure of an all pole system

ARMA Lattice-Ladder Filters:

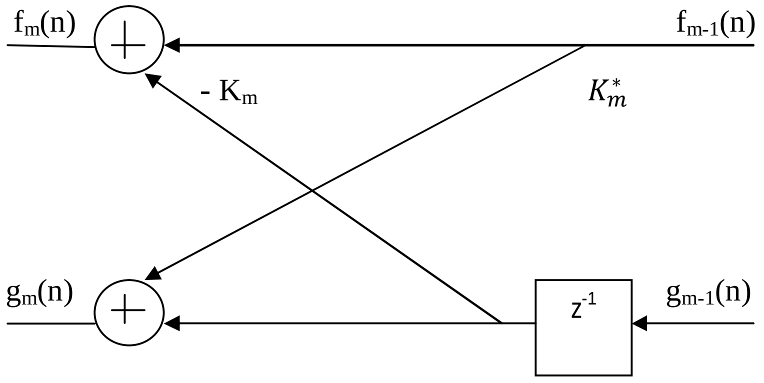
We consider an all-pole lattice filter with coefficients K_m , $1 \leq m \leq p$, and we add a ladder part by taking as the output a weighted linear combination of $\{g_m(n)\}$.The result is a pole-zero filter that has the lattice-ladder structure. Its output is

$$y(n) = \sum_{k=0}^{q} \beta_k\, g_k(n)$$

where $\{\beta_k\}$ are the parameters that determine the zeros of the system.



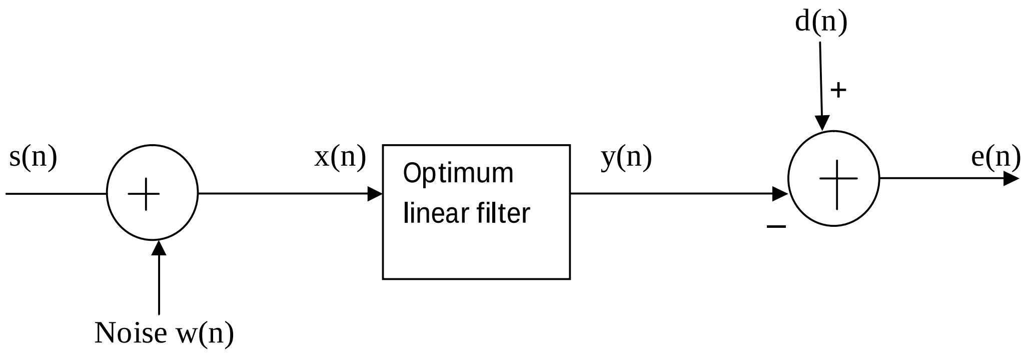
Pole-zero system



m^th stage lattice

Wiener filter for filtering and prediction:

It deals with the problem of estimating a desired signal {*s(n)}* in the presence of an undesired additive noise disturbance {*w(n)}*. The estimator is constrained to be a linear filter with impulse response {h(n)}, designed so that the output approximates some specified desired signal sequence {d(n)}.



Weiner Filter

The criterion selected for optimizing the filter impulse response {h(n)} is the minimization of the mean-square error. The optimum linear filter, in the sense of minimum mean-square error (MMSE), is called a Wiener filter.

Orthogonality Principle in Linear Mean-square Estimation

The mean-square error ε_M is a minimum if the filter coefficients {h(k)} are selected such that the error is orthogonal to each of the data points in estimate,

$$E[e(n)\, x^*(n-l)] = 0, \quad l = 0, 1, \ldots, M-1$$

where

$$e(n) = d(n) - \sum_{k=0}^{M-1} h(k)x(n-k)$$

Since the MSE is minimized by selecting the filter coefficients to satisfy the orthogonality principle, the residual minimum MSE is simply

$$MMSE_M = E[e(n)\, d^*(n)]$$

Module 3.

Power spectrum estimation

Power Spectrum Estimation

The power spectrum estimation deals with the estimation of the spectral characteristics of signals characterized as random processes. Many of the phenomena that occur in nature are best characterized statistically in terms of averages. For example, meteorological phenomena such as the fluctuations in air temperature and pressure are best characterized statistically as random processes.

Due to random fluctuations in such signals, we must adopt a statistical view point, which deals with the average characteristics of random signals. In particular, the autocorrelation function of random process is the appropriate statistical average that we will use for characterizing random signals in the time domain, and the Fourier transform of the autocorrelation function, which yields the power density spectrum, provides the transform from the time domain to frequency domain.

Estimation of Spectra from Finite-Duration Observation of Signals:

Computation of the Energy Density Spectrum:

Let $x(n)$ be a finite-duration sequence obtained by sampling a continuous-time signal $x_a(t)$ at some uniform sampling rate F_s.

If x(t) is a finite-energy signal, that is

$$E = \int_{-\infty}^{\infty} |x_a(t)|^2 dt < \infty$$

then its Fourier transform exists and is given as

$$X_a(F) = \int_{-\infty}^{\infty} x_a(t) e^{-j2\pi Ft}\, dt$$

From Parseval’s theorem we have

$$E = \int_{-\infty}^{\infty} |x_a(t)|^2 dt = \int_{-\infty}^{\infty} |X_a(F)|^2 dF$$

The quantity $|X_a(F)|^2$ ***represents the distribution of signal energy as a function of frequency****,* ***and hence it is called the energy density spectrum of the signal***, that is,

$$S_{xx}(F) = |X_a(F)|^2$$

Thus the total energy in the signal is simply the integral of $S_{xx}(F)$ over all F [i.e., the total area under $S_{xx}(F)$].

$S_{xx}(F)$ can be viewed as the Fourier transform of another function, $R_{xx}(\tau)$, called the autocorrelation function of the finite-energy signal $x_a(t)$, defined as

$$R_{xx}(\tau) = \int_{-\infty}^{\infty} x_a^*(t)\, x_a(t+\tau) dt$$

It follows that

$$\int_{-\infty}^{\infty} R_{xx}(\tau)\, e^{-j2\pi F\tau} d\tau = S_{xx}(F) = |X_a(F)|^2$$

Hence $R_{xx}(\tau)$ and $S_{xx}(F)$ are a Fourier transform pair.

Now the Fourier transform (voltage spectrum) of $x(n)$:

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

or, equivalently,

$$X(f) = \sum_{n=-\infty}^{\infty} x(n) e^{-j2\pi f n}$$

or,

$$X\left(\frac{F}{X(F)}\right) = F_s \sum_{k=-\infty}^{\infty} X_a(F - kF_s)$$

where $f = \frac{F}{F_s}$ is the normalized frequency variable.

In absence of aliasing, within the fundamental range $|F| \leq \frac{F_s}{2}$, we have

$$X\left(\frac{F}{X(F)}\right) = F_s X_a(F) \;, |F| \leq \frac{F_s}{2}$$

Hence the voltage spectrum of the sampled signal is identical to the voltage spectrum of the analog signal. As a consequence, the ***energy density spectrum of the sampled signal*** is

$$S_{xx}\left(\frac{F}{S_{xx}(F)}\right) = \left|X\left(\frac{F}{X(F)}\right)\right|^2 = F_s^2|X_a(F)|^2$$

The autocorrelation of the sampled signal $x(n)$ is defined as:

$$r_{xx}(k) = \sum_{n=-\infty}^{\infty} x^*(n)x(n+k)$$

its Fourier transform (Wiener-Khintchine theorem):

$$S_{xx}(f) = \sum_{n=-\infty}^{\infty} r_{xx}(k)\, e^{-j2\pi kf}$$

Hence the ***energy density spectrum can be obtained by the Fourier transform of the autocorrelation of the sequence*** $\{\boldsymbol{x(n)}\}$, that is,

$$S_{xx}(f) = |X(f)|^2$$
$$= \left|\sum_{n=-\infty}^{\infty} x(n)e^{-j2\pi fn}\right|^2$$

Estimation of the Autocorrelation and Power Spectrum of Random Signals: The Periodogram:

The finite-energy signals possess a Fourier transform and are characterized in the spectral domain by their energy density spectrum. On the other hand, the important class of signals characterized as stationary random processes do not have finite energy and hence do not posses a Fourier transform. Such signals have finite average power and hence are characterized by a power density spectrum.

If $x(t)$ is a stationary random process, its autocorrelation function is

$$\gamma_{xx}(\tau) = E[x^*(t)x(t+\tau)]$$

where $E[.]$ denotes the statistical average. Then by Wiener-Khintchine theorem, the power density spectrum of the stationary random process is the Fourier transform of the autocorrelation function:

$$\Gamma_{xx}(F) = \int_{-\infty}^{\infty} \gamma_{xx}(\tau) e^{-j2\pi F\tau} dt$$

But we do not know the true autocorrelation function $\gamma_{xx}(\tau)$ and as a consequence, we cannot compute the Fourier transform in (1.13) to obtain $\Gamma_{xx}(F)$. On the other hand, from a single realization of the random process we can compute the ***time-average autocorrelation function***:

$$R_{xx}(\tau) = \frac{1}{2T_0} \int_{-T_0}^{T_0} x^*(t) x(t+\tau) d\tau$$

where $2T_0$ is the observation interval.

The Fourier transform of $R_{xx}(\tau)$ provides an estimate $P_{xx}(F)$ of the power density spectrum, that is,

$$P_{xx}(F) = \int_{-T_0}^{T_0} R_{xx}(\tau) e^{-j2\pi F\tau} d\tau$$

$$= \frac{1}{2T_0} \int_{-T_0}^{T_0} \left[\int_{-T_0}^{T_0} x^*(t) x(t+\tau) d\tau \right] e^{-j2\pi F\tau} d\tau$$

$$= \frac{1}{2T_0} \left| \int_{-T_0}^{T_0} x(t) e^{-j2\pi Ft} dt \right|^2$$

The actual power density spectrum is the expected value of $P_{xx}(F)$ in the limit as $T_0 \to \infty$,

$$\Gamma_{xx}(F) = \lim_{T_0 \to \infty} E[P_{xx}(F)]$$

$$= \lim_{T_0 \to \infty} E\left[\frac{1}{2T_0} \left| \int_{-T_0}^{T_0} x(t) e^{-j2\pi Ft} dt \right|^2 \right]$$

The estimate $P_{xx}(F)$ can also be expressed as

$$P_{xx}(F) = \frac{1}{N} \left| \sum_{n=0}^{N-1} x(n) e^{-j2\pi fn} \right|^2 = \frac{1}{N} |X(f)|^2$$

where $X(f)$ is the Fourier transform of the finite duration sequence $x(n)$, $0 \le n \le N-1$. This form of the power density spectrum estimate is called the ***periodogram***.

Nonparametric Methods for Power Spectrum Estimation:

The nonparametric methods make no assumption about how the data were generated.

The Bartlett Method: Averaging Periodograms:

It reduces the variance in the periodogram. The N-point sequence is subdivided into K nonoverlapping segments, where each segment has length M. This results in the K data segments

$$x_i(n) = x(n + iM), \qquad \begin{matrix} i = 0, 1, \dots, K-1 \\ n = 0, 1, \dots, M-1 \end{matrix}$$

For each segment, we compute the periodogram

$$P_{xx}^{(i)}(f) = \frac{1}{M}\left|\sum_{n=0}^{M-1} x_i(n) e^{-j2\pi f n}\right|^2, \qquad i = 0, 1, \dots, K-1$$

The Bartlett power spectrum estimate obtained by averaging the periodograms for the K segments is

$$P_{xx}^{B}(f) = \frac{1}{K}\sum_{i=0}^{K-1} P_{xx}^{(i)}(f)$$

The mean value:

$$E[P_{xx}^{B}(f)] = \frac{1}{K}\sum_{i=0}^{K-1} E[P_{xx}^{(i)}(f)]$$
$$= E\left[P_{xx}^{(i)}(f)\right]$$

The expected value of single periodogram:

$$E\left[P_{xx}^{(i)}(f)\right] = \sum_{m=-(M-1)}^{M-1}\left(1 - \frac{|m|}{M}\right)\gamma_{xx}(m) e^{-j2\pi f m}$$

$$= \frac{1}{M}\int_{-\frac{1}{2}}^{\frac{1}{2}} \Gamma_{xx}(\alpha)\left(\frac{\sin \pi(f-\alpha) M}{\sin \pi(f-\alpha)}\right)^2 d\alpha$$

where

$$W_B(f) = \frac{1}{M}\left(\frac{\sin \pi f M}{\sin \pi f}\right)^2$$

is the frequency characteristic of the Bartlett window

$$w_B(n) = \begin{cases} 1 - \frac{|m|}{M}, & |m| \le M - 1 \\ 0, & otherwise \end{cases}$$

The variance of the Bartlett estimate:

$$var[P_{xx}^B(f)] = \frac{1}{K^2}\sum_{i=0}^{K-1} var[P_{xx}^{(i)}(f)]$$

$$= \frac{1}{K} var\left[P_{xx}^{(i)}(f)\right]$$

But

$$var[P_{xx}(f)] = \Gamma_{xx}^2(f)\left[1 + \left(\frac{\sin 2\pi f N}{N \sin 2\pi f}\right)^2\right]$$

Putting the above value in (2.8), we get

$$var[P_{xx}^B(f)] = \frac{1}{K^2}\Gamma_{xx}^2(f)\left[1 + \left(\frac{\sin 2\pi f N}{N \sin 2\pi f}\right)^2\right]$$

Therefore, the variance of the Bartlett power spectrum estimate has been reduced by the factor K.

The Blackman and Tukey Method: Smoothing the Periodogram:

In this method the sample autocorrelation sequence is windowed first and then Fourier transformed to yield the estimate of the power spectrum.

The Blackman-Tukey estimate is

$$P_{xx}^{BT}(f) = \sum_{m=-(M-1)}^{M-1} r_{xx}(m)w(m)e^{-j2\pi f m}$$

where the window function $w(n)$ has length $2M - 1$ and is zero for $|m| \ge M$.

The frequency domain equivalent expression:

$$P_{xx}^{BT}(f) = \int_{-1/2}^{1/2} P_{xx}(\alpha)W(f-\alpha)d\alpha$$

where $P_{xx}(f)$ is the periodogram.

The expected value of the Blackman-Tukey power spectrum estimate:

$$E[P_{xx}^{BT}(f)] = \int_{-1/2}^{1/2} E[P_{xx}(\alpha)]W(f-\alpha)d\alpha$$

where

$$E[P_{xx}(\alpha)] = \int_{-1/2}^{1/2} \Gamma_{xx}(\theta)W_B(\alpha-\theta)d\theta$$

and $W_B(f)$ is the Fourier transform of the Bartlett window. Substitution of (2.14) into (2.13) yields

$$E[P_{xx}^{BT}(f)] = \int_{-1/2}^{1/2}\int_{-1/2}^{1/2} \Gamma_{xx}(\theta)W_B(\alpha-\theta)\,W(f-\alpha)d\alpha d\theta$$

In time domain we have:

$$E[P_{xx}^{BT}(f)] = \sum_{m=-(M-1)}^{M-1} E[r_{xx}(m)]w(m)\,e^{-j2\pi fm}$$

$$= \sum_{m=-(M-1)}^{M-1} \gamma_{xx}(m)w_B(m)w(m)\,e^{-j2\pi fm}$$

where the Bartlett window is,

$$w_B(m) = \begin{cases} 1-\dfrac{|m|}{N}\ , & |m| < N \\ 0\ , & otherwise \end{cases}$$

The variance of the Blackman-Tukey power spectrum estimate is

$$var[P_{xx}^{BT}(f)] = E\{[P_{xx}^{BT}(f)]^2\} - \{E[P_{xx}^{BT}(f)]\}^2$$

Assuming $W(f)$ is narrow compared to the true power spectrum $\Gamma_{xx}(f)$

$$var[P_{xx}^{BT}(f)] \approx \Gamma_{xx}^2(f)\left[\frac{1}{N}\int_{-1/2}^{1/2} W^2(\theta)d\theta\right]$$

$$\approx \Gamma_{xx}^2(f)\left[\frac{1}{N}\sum_{m=-(M-1)}^{M-1} w^2(m)\right]$$

<u>Parametric Methods for Power Spectrum Estimation</u>:

Parametric methods avoid the problem of spectral leakage and provide better frequency resolution than do the nonparametric methods. Parametric methods also eliminate the need for window functions.

The Yuke-Walker Method for the AR Model Parameters:

This method is used to estimate the autocorrelation from the data and use the estimates to solve for the AR model parameters.
The autocorrelation estimate is given by

$$r_{xx}(m) = \frac{1}{N}\sum_{n=0}^{N-m-1} x^*(n)x(n+m), \qquad m \geq 0$$

The corresponding power spectrum estimate is

$$P_{xx}^{YW}(f) = \frac{\hat{\sigma}_{wp}^2}{\left|1 + \sum_{k=1}^{p} \hat{a}_p(k)e^{-j2\pi fk}\right|^2}$$

where $\hat{a}_p(k)$ are estimates of the AR parameters and

$$\hat{\sigma}_{wp}^2 = \hat{E}_p^f = r_{xx}(0)\prod_{k=1}^{p}[1 - |\hat{a}_k(k)|^2]$$

is the estimated minimum mean-square value for the pth-order predictor.

The Burg method for the AR Model Parameters:

Suppose that we are given the data $x(n)$, $n = 0,1, \dots, N-1$. The forward and backward linear prediction estimates of order m are given as:

$$\hat{x}(n) = -\sum_{k=1}^{m} a_m(k)x(n-k)$$

$$\hat{x}(n-m) = -\sum_{k=1}^{m} a_m^*(k)x(n+k-m)$$

and the corresponding forward and backward errors $f_m(n)$ and $g_m(n)$ defined as $f_m(n) = x(n) - \hat{x}(n)$ and $g_m(n) = x(n-m) - \hat{x}(n-m)$ where $a_m(k), 0 \leq k \leq m-1$, $m = 1, 2, \dots, p$, are prediction coefficients.

The least-squares error is

$$\varepsilon_m = \sum_{n=m}^{N-1} [|f_m(n)|^2 + |g_m(n)|^2]$$

The power spectrum estimate is

$$P_{xx}^{BU}(f) = \frac{\hat{E}_p}{\left|1 + \sum_{k=1}^{p} \hat{a}_p(k)e^{-j2\pi fk}\right|^2}$$

where $\hat{E}_p$ is the total least-squares error.

Advantages of the Burg method:

- results in high frequency resolution
- yields a stable AR model
- computationally efficient

Disadvantages of the Burg method:

- exhibits spectral line splitting at high signal-to-noise ratios
- exhibits a sensitivity to the initial phase of sinusoid for sinusoidal signals in noise resulting in a phase-dependent frequency bias.

The MA Model for Power Spectrum Estimation :

The parameters of MA model are related to the statistical autocorrelation $\gamma_{xx}(m)$ by

$$\gamma_{xx}(m) = \begin{cases} \sigma_w^2 \sum_{k=0}^{q} b_k b_{k+m}, & 0 \le m \le q \\ 0, & m > q \\ \gamma_{xx}^*(-m), & m < 0 \end{cases}$$

However,

$$B(z)B(z^{-1}) = D(z) = \sum_{m=-q}^{q} d_m z^{-m}$$

where coefficients $\{d_m\}$ are related to the MA parameters by the expression

$$d_m = \sum_{k=0}^{q-|m|} b_k b_{k+m}, \qquad |m| \le q$$

Clearly, then,

$$\gamma_{xx}(m) = \begin{cases} \sigma_w^2 d_m, & |m| \le q \\ 0, & |m| > q \end{cases}$$

and the power spectrum for MA process is

$$\Gamma_{xx}^{MA}(f) = \sum_{m=-q}^{q} \gamma_{xx}(m) e^{-j2\pi fn}$$

It is apparent from these expressions that we do not have to solve for the MA parameters $\{b_k\}$ to estimate the power spectrum. The estimates of the autocorrelation $\gamma_{xx}(m)$ for $|m| \le q$ suffice. From such estimates we compute the estimated MA power spectrum , given as

$$\mathrm{P}_{xx}^{MA}(f) = \sum_{m=-q}^{q} r_{xx}(m) e^{-j2\pi fm}$$

The ARMA Model for Power Spectrum Estimation :

An ARMA model provides us with an opportunity to improve on the AR spectrum estimate by using few model parameters. The ARMA model is particularly appropriate when the signal has been corrupted by noise.

The sequence $x(n)$ can be filtered by an FIR filter to yield the sequence

$$v(n) = x(n) + \sum_{k=1}^{p} \hat{a}_k x(n-k) \ , \qquad n = 0, 1, \dots , N-1$$

The filtered sequence $v(n)$ for $p \leq n \leq N-1$ is used to form the estimated correlation sequences $r_{vv}(m)$, from which we obtain the MA spectrum

$$\mathrm{P}_{vv}^{MA}(f) = \sum_{m=-q}^{q} r_{vv}(m) e^{-j2\pi f m}$$

Finally, the estimated ARMA power spectrum is

$$\hat{P}_{xx}^{ARMA}(f) = \frac{\mathrm{P}_{vv}^{MA}(f)}{\left|1 + \sum_{k=1}^{p} \hat{a}_k e^{-j2\pi f k}\right|^2}$$

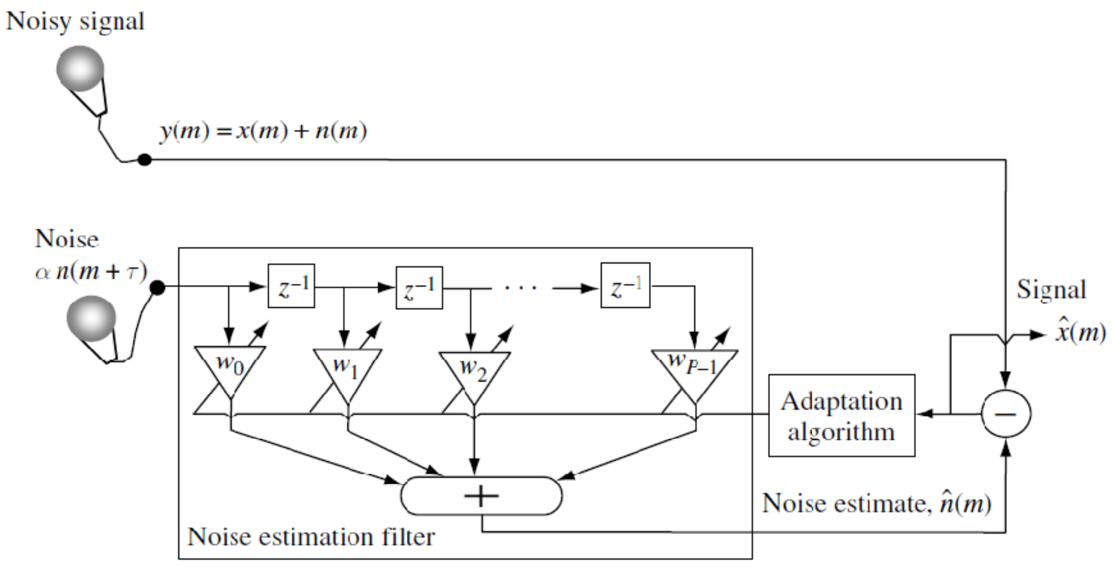
Module 4.

Adaptive Signal Processing

ADAPTIVE NOISE CANCELLATION

In speech communication from a noisy acoustic environment such as a moving car or train, or over a noisy telephone channel, the speech signal is observed in an additive random noise.

In signal measurement systems the information-bearing signal is often contaminated by noise from its surrounding environment. The noisy observation, y(m), can be modelled as
y(m) = x(m)+n(m)
where x(m) and n(m) are the signal and the noise, and m is the discrete-time index. In some situations, for example when using a mobile telephone in a moving car, or when using a radio communication device in an aircraft cockpit, it may be possible to measure and estimate the instantaneous amplitude of the ambient noise using a directional microphone. The signal, x(m), may then be recovered by subtraction of an estimate of the noise from the noisy signal.

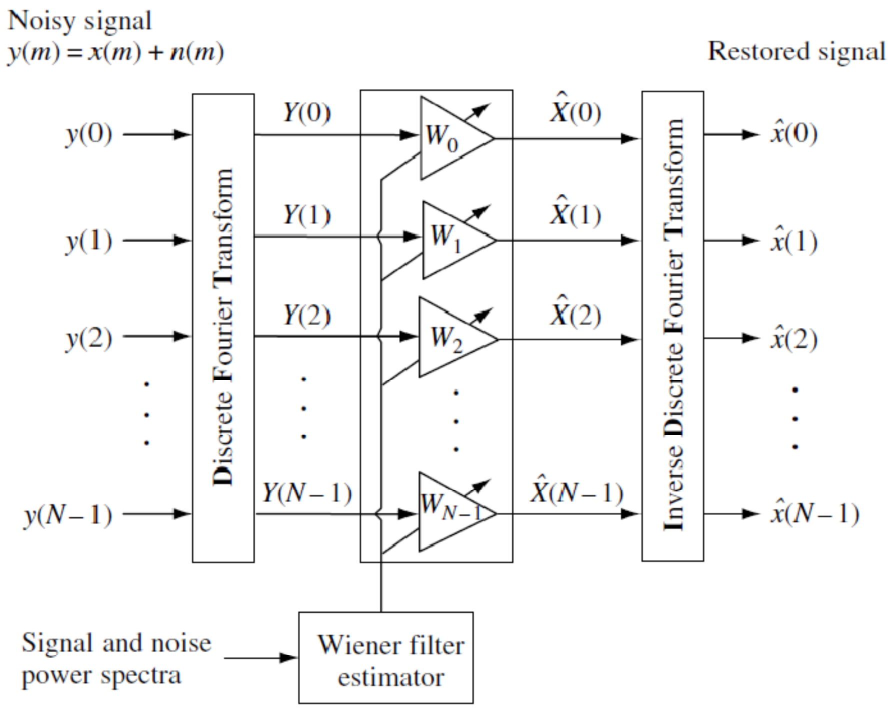


(Configuration of a two-microphone adaptive noise canceller.)

Figure shows a two-input adaptive noise cancellation system for enhancement of noisy speech. In this system a directional microphone takes as input the noisy signal x(m)+n(m), and a second directional microphone, positioned some distance away, measures the noise αn(m+ζ). The attenuation factor α and the time delay ζ provide a rather over-simplified model of the effects of propagation of the noise to different positions in the space where the microphones are placed. The noise from the second microphone is processed by an adaptive digital filter to make it equal to the noise contaminating the speech signal, and then subtracted from the noisy signal to cancel out the noise. The adaptive noise canceller is more effective in cancelling out the low-frequency part of the noise, but generally suffers from the non-stationary character of the signals, and from the over-simplified assumption that a linear filter can model the diffusion and propagation of the noise sound in the space.

ADAPTIVE NOISE REDUCTION

In many applications, for example at the receiver of a telecommunication system, there is no access to the instantaneous value of the contaminating noise, and only the noisy signal is available. In such cases the noise cannot be cancelled out, but it may be reduced, in an average sense, using the statistics of the signal and the noise process.



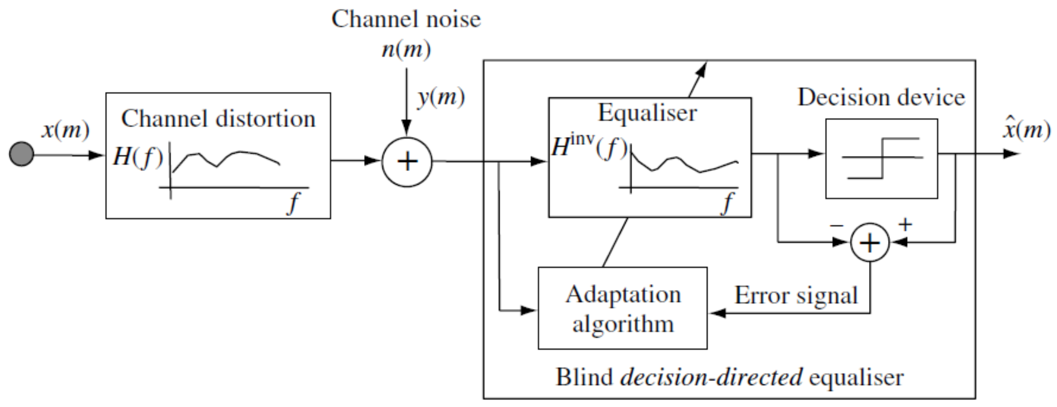
A frequency-domain Wiener filter for reducing additive noise

Figure shows a bank of Wiener filters for reducing additive noise when only the noisy signal is available. The filter bank coefficients attenuate each noisy signal frequency in inverse proportion to

the signal-to-noise ratio at that frequency. The Wiener filter bank coefficients, are calculated from estimates of the power spectra of the signal and the noise processes.

BLIND CHANNEL EQUALISATION

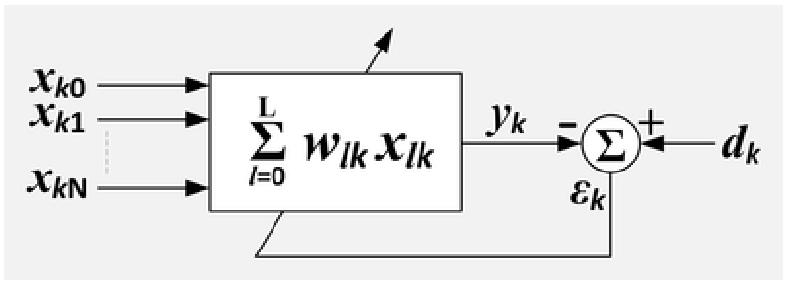
Channel equalisation is the recovery of a signal distorted in transmission through a communication channel with a nonflat magnitude or a nonlinear phase response. When the channel response is unknown, the process of signal recovery is called 'blind equalisation'. Blind equalisation has a wide range of applications, for example in digital telecommunications for removal of inter-symbol interference due to nonideal channel and multipath propagation, in speech recognition for removal of the effects of the microphones and communication channels, in correction of distorted images, in analysis of seismic data and in de-reverberation of acoustic gramophone recordings. In practice, blind equalisation is feasible only if some useful statistics of the channel input are available. The success of a blind equalisation method depends on how much is known about the characteristics of the input signal and how useful this knowledge can be in the channel identification and equalisation process. Figure 1.6 illustrates the configuration of a decision-directed equaliser. This blind channel equaliser is composed of two distinct sections: an adaptive equaliser that removes a large part of the channel distortion, followed by a nonlinear decision device for an improved estimate of the channel input. The output of the decision device is the final estimate of the channel input, and it is used as the desired signal to direct the equaliser adaptation process.



Configuration of a decision-directed blind channel equaliser.

Adaptive Linear Combiner

Adaptive linear combiner showing the combiner and the adaption process. k = sample number, n=input variable index, x = reference inputs, d = desired input, W = set of filter coefficients, ε = error output, Σ = summation, upper box=linear combiner, lower box=adaption algorithm.



Adaptive linear combiner, compact representation. k = sample number, n=input variable index, x = reference inputs, d = desired input, ε = error output, Σ = summation.

The adaptive linear combiner (ALC) resembles the adaptive tapped delay line FIR filter except that there is no assumed relationship between the X values. If the X values were from the outputs of a tapped delay line, then the combination of tapped delay line and ALC would comprise an adaptive filter. However, the X values could be the values of an array of pixels. Or they could be the outputs of multiple tapped delay lines. The ALC finds use as an adaptive beam former for arrays of hydrophones or antennas.

$$y_k = \sum_{l=0}^{L} w_{lk}\ x_{lk}$$

where w_{lk} refers to the l'th weight at k'th time.

LMS algorithm

Main article: Least mean squares filter

If the variable filter has a tapped delay line FIR structure, then the LMS update algorithm is especially simple. Typically, after each sample, the coefficients of the FIR filter are adjusted as follows:[5](Widrow)

for $l = 0 \ldots L$

$w_{l,k+1} = w_{lk} + 2\mu\ \epsilon_k\ x_{k-l}$

μ is called the convergence factor.

The LMS algorithm does not require that the X values have any particular relationship; therefor it can be used to adapt a linear combiner as well as an FIR filter. In this case the update formula is written as:

$$w_{l,k+1} = w_{lk} + 2\mu\ \epsilon_k\ x_{lk}$$

The effect of the LMS algorithm is at each time, k, to make a small change in each weight. The direction of the change is such that it would decrease the error if it had been applied at time k. The magnitude of the change in each weight depends on μ, the associated X value and the error at time k. The weights making the largest contribution to the output, y_k, are changed the most. If the error is zero, then there should be no change in the weights. If the associated value of X is zero, then changing the weight makes no difference, so it is not changed.

Convergence

μ controls how fast and how well the algorithm converges to the optimum filter coefficients. If μ is too large, the algorithm will not converge. If μ is too small the algorithm converges slowly and may not be able to track changing conditions. If μ is large but not too large to prevent convergence, the algorithm reaches steady state rapidly but continuously overshoots the optimum weight vector. Sometimes, μ is made large at first for rapid convergence and then decreased to minimize overshoot.

Widrow and Stearns state in 1985 that they have no knowledge of a proof that the LMS algorithm will converge in all cases.

However under certain assumptions about stationarity and independence it can be shown that the algorithm will converge if

$$0 < \mu < \frac{1}{\sigma^2}$$

where

$$\sigma^2 = \sum_{l=0}^{L} \sigma_l^2$$ = sum of all input power

σ_l is the RMS value of the l'th input

In the case of the tapped delay line filter, each input has the same RMS value because they are simply the same values delayed. In this case the total power is

$$\sigma^2 = (L+1)\sigma_0^2$$

where

σ_0 is the RMS value of x_k, the input stream.

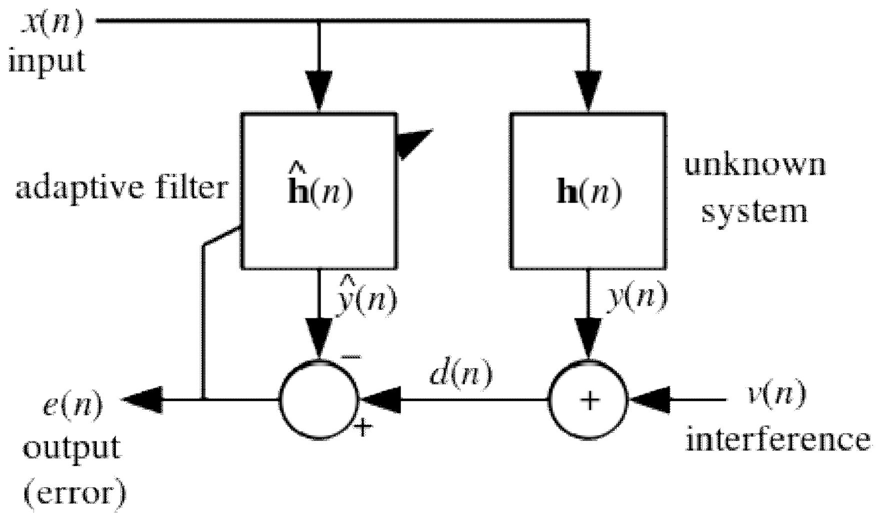
This leads to a normalized LMS algorithm:

$w_{l,k+1} = w_{lk} + \left(\frac{2\mu_\sigma}{\sigma^2}\right) \epsilon_k \; x_{k-l}$ in which case the convergence criteria becomes: $0 < \mu_\sigma < 1$.

Least mean squares filter

Least mean squares (LMS) algorithms are a class of adaptive filter used to mimic a desired filter by finding the filter coefficients that relate to producing the least mean squares of the error signal (difference between the desired and the actual signal). It is a stochastic gradient descent method in that the filter is only adapted based on the error at the current time. It was invented in 1960 by Stanford University professor Bernard Widrow and his first Ph.D. student, Ted Hoff.

Problem formulation



Relationship to the least mean squares filter

The realization of the causal Wiener filter looks a lot like the solution to the least squares estimate, except in the signal processing domain. The least squares solution, for input matrix $\mathbf{X}$ and output vector $\boldsymbol{y}$ is

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathbf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathbf{T}}\boldsymbol{y}.$$

The FIR least mean squares filter is related to the Wiener filter, but minimizing the error criterion of the former does not rely on cross-correlations or auto-correlations. Its solution converges to the Wiener filter solution. Most linear adaptive filtering problems can be formulated using the block diagram above. That is, an unknown system $\mathbf{h}(n)$ is to be identified and the adaptive filter attempts to adapt the filter $\hat{\mathbf{h}}(n)$ to make it as close as possible to $\mathbf{h}(n)$, while using only observable signals $x(n)$, $d(n)$ and $e(n)$; but $y(n)$, $v(n)$ and $h(n)$ are not directly observable. Its solution is closely related to the Wiener filter.

Definition of symbols

n is the number of the current input sample

p is the number of filter taps

$\{\cdot\}^H$ (Hermitian transpose or conjugate transpose)

$\mathbf{x}(n) = [x(n), x(n-1), \ldots, x(n-p+1)]^T$

$$\mathbf{h}(n) = [h_0(n), h_1(n), \ldots, h_{p-1}(n)]^T, \quad \mathbf{h}(n) \in \mathbb{C}^p$$

$$y(n) = \mathbf{h}^H(n) \cdot \mathbf{x}(n)$$

$$d(n) = y(n) + \nu(n)$$

$\hat{\mathbf{h}}(n)$ estimated filter; interpret as the estimation of the filter coefficients after n samples

$$e(n) = d(n) - \hat{y}(n) = d(n) - \hat{\mathbf{h}}^H(n) \cdot \mathbf{x}(n)$$

Idea

The basic idea behind LMS filter is to approach the optimum filter weights $(R^{-1}P)$, by updating the filter weights in a manner to converge to the optimum filter weight. The algorithm starts by assuming a small weights (zero in most cases), and at each step, by finding the gradient of the mean square error, the weights are updated. That is, if the MSE-gradient is positive, it implies, the error would keep increasing positively, if the same weight is used for further iterations, which means we need to reduce the weights. In the same way, if the gradient is negative, we need to increase the weights. So, the basic weight update equation is :

$$W_{n+1} = W_n - \mu \nabla \varepsilon[n],$$

where ε represents the mean-square error. The negative sign indicates that, we need to change the weights in a direction opposite to that of the gradient slope.

The mean-square error, as a function of filter weights is a quadratic function which means it has only one extrema, that minimises the mean-square error, which is the optimal weight. The LMS thus, approaches towards this optimal weights by ascending/descending down the mean-square-error vs filter weight curve.

Derivation

The idea behind LMS filters is to use steepest descent to find filter weights $\mathbf{h}(n)$ which minimize a cost function. We start by defining the cost function as

$$C(n) = E\left\{|e(n)|^2\right\}$$

where $e(n)$ is the error at the current sample n and $E\{\cdot\}$ denotes the expected value.

This cost function ($C(n)$) is the mean square error, and it is minimized by the LMS. This is where the LMS gets its name. Applying steepest descent means to take the partial derivatives with respect to the individual entries of the filter coefficient (weight) vector

$$\nabla_{\hat{\mathbf{h}}^H} C(n) = \nabla_{\hat{\mathbf{h}}^H} E\left\{e(n)\,e^*(n)\right\} = 2E\left\{\nabla_{\hat{\mathbf{h}}^H}(e(n))e^*(n)\right\}$$

where $\nabla$is the gradient operator

$$\nabla_{\hat{\mathbf{h}}^H}(e(n)) = \nabla_{\hat{\mathbf{h}}^H}\left(d(n) - \hat{\mathbf{h}}^H \cdot \mathbf{x}(n)\right) = -\mathbf{x}(n)$$

$$\nabla C(n) = -2E\left\{\mathbf{x}(n)\,e^*(n)\right\}$$

Now, $\nabla C(n)$is a vector which 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 of $\nabla C(n)$. To express that in mathematical terms

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) - \frac{\mu}{2}\nabla C(n) = \hat{\mathbf{h}}(n) + \mu\, E\left\{\mathbf{x}(n)\,e^*(n)\right\}$$

where $\frac{\mu}{2}$is the step size(adaptation constant). That means we have found a sequential update algorithm which minimizes the cost function. Unfortunately, this algorithm is not realizable until we know $E\left\{\mathbf{x}(n)\,e^*(n)\right\}$.

Generally, the expectation above is not computed. Instead, to run the LMS in an online (updating after each new sample is received) environment, we use an instantaneous estimate of that expectation. See below.

Simplifications

For most systems the expectation function $E\left\{\mathbf{x}(n)\,e^*(n)\right\}$must be approximated. This can be done with the following unbiased estimator

$$\hat{E}\left\{\mathbf{x}(n)\,e^*(n)\right\} = \frac{1}{N}\sum_{i=0}^{N-1}\mathbf{x}(n-i)\,e^*(n-i)$$

where Nindicates the number of samples we use for that estimate. The simplest case is $N = 1$

$$\hat{E}\left\{\mathbf{x}(n)\,e^*(n)\right\} = \mathbf{x}(n)\,e^*(n)$$

For that simple case the update algorithm follows as

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \mu \mathbf{x}(n) e^*(n)$$

Indeed this constitutes the update algorithm for the LMS filter.

LMS algorithm summary

The LMS algorithm for a pth order algorithm can be summarized as

Parameters: $p =$filter order

$\mu =$step size

Initialisation: $\hat{\mathbf{h}}(0) = \mathrm{zeros}(p)$

Computation: For $n = 0, 1, 2, ...$

$$\mathbf{x}(n) = [x(n), x(n-1), \dots, x(n-p+1)]^T$$
$$e(n) = d(n) - \hat{\mathbf{h}}^H(n)\mathbf{x}(n)$$

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \mu\, e^*(n)\mathbf{x}(n)$$

Convergence and stability in the mean

As the LMS algorithm does not use the exact values of the expectations, the weights would never reach the optimal weights in the absolute sense, but a convergence is possible in mean. That is, even though the weights may change by small amounts, it changes about the optimal weights. However, if the variance with which the weights change, is large, convergence in mean would be misleading. This problem may occur, if the value of step-size $\mu$is not chosen properly.

If $\mu$is chosen to be large, the amount with which the weights change depends heavily on the gradient estimate, and so the weights may change by a large value so that gradient which was negative at the first instant may now become positive. And at the second instant, the weight may change in the opposite direction by a large amount because of the negative gradient and would thus keep oscillating with a large variance about the optimal weights. On the other hand if $\mu$is chosen to be too small, time to converge to the optimal weights will be too large.

Thus, an upper bound on $\mu$is needed which is given as $0 < \mu < \frac{2}{\lambda_{\mathrm{max}}}$

where $\lambda_{\mathbf{max}}$is the greatest eigenvalue of the autocorrelation matrix $\mathbf{R} = E\{\mathbf{x}(n)\mathbf{x}^H(n)\}$. If this condition is not fulfilled, the algorithm becomes unstable and $\hat{h}(n)$diverges.

Maximum convergence speed is achieved when

$$\mu = \frac{2}{\lambda_{\mathbf{max}} + \lambda_{\mathbf{min}}},$$

where $\lambda_{\mathbf{min}}$is the smallest eigenvalue of R. Given that $\mu$is less than or equal to this optimum, the convergence speed is determined by $\lambda_{\mathbf{min}}$, with a larger value yielding faster convergence. This means that faster convergence can be achieved when $\lambda_{\mathbf{max}}$is close to $\lambda_{\mathbf{min}}$, that is, the maximum achievable convergence speed depends on the eigenvalue spread of $\mathbf{R}$.

A white noise signal has autocorrelation matrix $\mathbf{R} = \sigma^2\mathbf{I}$where $\sigma^2$is the variance of the signal. In this case all eigenvalues are equal, and the eigenvalue spread is the minimum over all possible matrices. The common interpretation of this result is therefore that the LMS converges quickly for white input signals, and slowly for colored input signals, such as processes with low-pass or high-pass characteristics.

It is important to note that the above upperbound on μonly enforces stability in the mean, but the coefficients of $\hat{h}(n)$can still grow infinitely large, i.e. divergence of the coefficients is still possible. A more practical bound is

$$0 < \mu < \frac{2}{\mathrm{tr}\left[\mathbf{R}\right]},$$

where $\mathrm{tr}[\mathbf{R}]$denotes the trace of $\mathbf{R}$. This bound guarantees that the coefficients of $\hat{h}(n)$do not diverge (in practice, the value of μshould not be chosen close to this upper bound, since it is somewhat optimistic due to approximations and assumptions made in the derivation of the bound).

Normalised least mean squares filter (NLMS)

The main drawback of the "pure" LMS algorithm is that it is sensitive to the scaling of its input $x(n)$. This makes it very hard (if not impossible) to choose a learning rate μthat guarantees stability of the algorithm (Haykin 2002). The Normalised least mean squares filter (NLMS) is a variant of the LMS algorithm that solves this problem by normalising with the power of the input. The NLMS algorithm can be summarised as:

Parameters: $p =$ filter order

$\mu =$ step size

Initialization: $\hat{\mathbf{h}}(0) = \text{zeros}(p)$

Computation: For $n = 0, 1, 2, \ldots$

$$\mathbf{x}(n) = [x(n), x(n-1), \ldots, x(n-p+1)]^T$$
$$e(n) = d(n) - \hat{\mathbf{h}}^H(n)\mathbf{x}(n)$$

$$\hat{\mathbf{h}}(n+1) = \hat{\mathbf{h}}(n) + \frac{\mu\, e^*(n)\mathbf{x}(n)}{\mathbf{x}^H(n)\mathbf{x}(n)}$$

Optimal learning rate

It can be shown that if there is no interference ($v(n) = 0$), then the optimal learning rate for the NLMS algorithm is

$$\mu_{opt} = 1$$

and is independent of the input $x(n)$and the real (unknown) impulse response $\mathbf{h}(n)$. In the general case with interference ($v(n) \neq 0$), the optimal learning rate is

$$\mu_{opt} = \frac{E\left[\left|y(n) - \hat{y}(n)\right|^2\right]}{E\left[|e(n)|^2\right]}$$

The results above assume that the signals $v(n)$and $x(n)$are uncorrelated to each other, which is generally the case in practice.

Proof

Let the filter misalignment be defined as $\Lambda(n) = \left|\mathbf{h}(n) - \hat{\mathbf{h}}(n)\right|^2$, we can derive the expected misalignment for the next sample as:

$$E\left[\Lambda(n+1)\right] = E\left[\left|\hat{\mathbf{h}}(n) + \frac{\mu\, e^*(n)\mathbf{x}(n)}{\mathbf{x}^H(n)\mathbf{x}(n)} - \mathbf{h}(n)\right|^2\right]$$

$$E\left[\Lambda(n+1)\right] = E\left[\left|\hat{\mathbf{h}}(n) + \frac{\mu\,\left(v^*(n) + y^*(n) - \hat{y}^*(n)\right)\mathbf{x}(n)}{\mathbf{x}^H(n)\mathbf{x}(n)} - \mathbf{h}(n)\right|^2\right]$$

Let $\delta = \hat{\mathbf{h}}(n) - \mathbf{h}(n)$and $r(n) = \hat{y}(n) - y(n)$

$$E\left[\Lambda(n+1)\right] = E\left[\left|\delta(n) - \frac{\mu\,\left(v(n) + r(n)\right)\mathbf{x}(n)}{\mathbf{x}^H(n)\mathbf{x}(n)}\right|^2\right]$$

$$E\left[\Lambda(n+1)\right] = E\left[\left(\delta(n) - \frac{\mu\,\left(v(n) + r(n)\right)\mathbf{x}(n)}{\mathbf{x}^H(n)\mathbf{x}(n)}\right)^H \left(\delta(n) - \frac{\mu\,\left(v(n) + r(n)\right.}{\mathbf{x}^H(n)\mathbf{x}(n}\right.\right.$$

Assuming independence, we have:

$$E\left[\Lambda(n+1)\right] = \Lambda(n) + E\left[\left(\frac{\mu\,\left(v(n) - r(n)\right)\mathbf{x}(n)}{\mathbf{x}^H(n)\mathbf{x}(n)}\right)^H \left(\frac{\mu\,\left(v(n) - r(n)\right)\mathbf{x}(n)}{\mathbf{x}^H(n)\mathbf{x}(n)}\right)\right.$$

$$E\left[\Lambda(n+1)\right] = \Lambda(n) + \frac{\mu^2 E\left[|e(n)|^2\right]}{\mathbf{x}^H(n)\mathbf{x}(n)} - \frac{2\mu E\left[|r(n)|^2\right]}{\mathbf{x}^H(n)\mathbf{x}(n)}$$

The optimal learning rate is found at $\frac{dE\left[\Lambda(n+1)\right]}{d\mu} = 0$, which leads to:

$$2\mu E\left[|e(n)|^2\right] - 2E\left[|r(n)|^2\right] = 0$$

$$\mu = \frac{E\left[|r(n)|^2\right]}{E\left[|e(n)|^2\right]}$$

Wiener filter

In signal processing, the **Wiener filter** is a filter used to produce an estimate of a desired or target random process by linear time-invariant filtering of an observed noisy process, assuming known stationary signal and noise spectra, and additive noise. The Wiener filter minimizes the mean square error between the estimated random process and the desired process..

Description

The goal of the Wiener filter is to compute a statistical estimate of an unknown signal using a related signal as an input and filtering that known signal to produce the estimate as an output. For example, the known signal might consist of an unknown signal of interest that has been corrupted by additive noise. The Wiener filter can be used to filter out the noise from the corrupted signal to provide an estimate of the underlying signal of interest. The Wiener filter is based on a statistical approach, and a more statistical account of the theory is given in the minimum mean-square error (MMSE) article.

Typical deterministic filters are designed for a desired frequency response. However, the design of the Wiener filter takes a different approach. One is assumed to have knowledge of the spectral properties of the original signal and the noise, and one seeks the linear time-invariant filter whose output would come as close to the original signal as possible. Wiener filters are characterized by the following:[1]

1. Assumption: signal and (additive) noise are stationary linear stochastic processes with known spectral characteristics or known autocorrelation and cross-correlation
2. Requirement: the filter must be physically realizable/causal (this requirement can be dropped, resulting in a non-causal solution)
3. Performance criterion: minimum mean-square error (MMSE)

This filter is frequently used in the process of deconvolution; for this application, see Wiener deconvolution.

Wiener filter solutions

The Wiener filter problem has solutions for three possible cases: one where a noncausal filter is acceptable (requiring an infinite amount of both past and future data), the case where a causal filter is desired (using an infinite amount of past data), and the finite impulse response (FIR) case where a finite amount of past data is used. The first case is simple to solve but is not suited for real-time applications. Wiener's main accomplishment was solving the case where the causality requirement is in effect, and in an appendix of Wiener's book Levinson gave the FIR solution.

Noncausal solution

$$G(s) = \frac{S_{x,s}(s)}{S_x(s)} e^{\alpha s}.$$

Where S are spectra. Provided that $g(t)$ is optimal, then the minimum mean-square error equation reduces to

$$E(e^2) = R_s(0) - \int_{-\infty}^{\infty} g(\tau) R_{x,s}(\tau + \alpha)\, d\tau,$$

and the solution $g(t)$ is the inverse two-sided Laplace transform of $G(s)$.

Causal solution

$$G(s) = \frac{H(s)}{S_x^+(s)},$$

where

- $H(s)$ consists of the causal part of $\frac{S_{x,s}(s)}{S_x^-(s)} e^{\alpha s}$ (that is, that part of this fraction having a positive time solution under the inverse Laplace transform)
- $S_x^+(s)$ is the causal component of $S_x(s)$ (i.e., the inverse Laplace transform of $S_x^+(s)$ is non-zero only for $t \geq 0$)
- $S_x^-(s)$ is the anti-causal component of $S_x(s)$ (i.e., the inverse Laplace transform of $S_x^-(s)$ is non-zero only for $t < 0$)

This general formula is complicated and deserves a more detailed explanation. To write down the solution $G(s)$ in a specific case, one should follow these steps:[2]

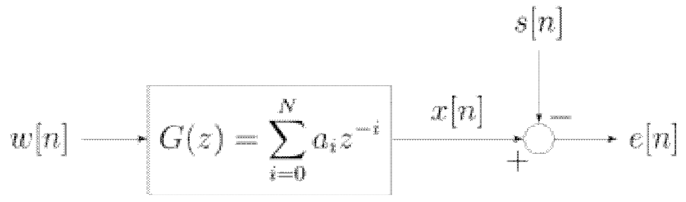
1. Start with the spectrum $S_x(s)$ in rational form and factor it into causal and anti-causal components:

$$S_x(s) = S_x^+(s) S_x^-(s)$$

where S^+ contains all the zeros and poles in the left half plane (LHP) and S^- contains the zeroes and poles in the right half plane (RHP). This is called the Wiener–Hopf factorization.

2. Divide $S_{x,s}(s)e^{\alpha s}$ by $S_x^-(s)$ and write out the result as a partial fraction expansion.
3. Select only those terms in this expansion having poles in the LHP. Call these terms $H(s)$.
4. Divide $H(s)$ by $S_x^+(s)$. The result is the desired filter transfer function $G(s)$.

Finite impulse response Wiener filter for discrete series



Block diagram view of the FIR Wiener filter for discrete series. An input signal w[n] is convolved with the Wiener filter g[n] and the result is compared to a reference signal s[n] to obtain the filtering error e[n].

The causal finite impulse response (FIR) Wiener filter, instead of using some given data matrix X and output vector Y, finds optimal tap weights by using the statistics of the input and output signals. It populates the input matrix X with estimates of the auto-correlation of the input signal (T) and populates the output vector Y with estimates of the cross-correlation between the output and input signals (V).

In order to derive the coefficients of the Wiener filter, consider the signal w[n] being fed to a Wiener filter of order N and with coefficients $\{a_i\}$, $i = 0, \ldots, N$. The output of the filter is denoted x[n] which is given by the expression

$$x[n] = \sum_{i=0}^{N} a_i w[n-i].$$

The residual error is denoted e[n] and is defined as e[n] = x[n] − s[n] (see the corresponding block diagram). The Wiener filter is designed so as to minimize the mean square error (MMSE criteria) which can be stated concisely as follows:

$$a_i = \arg\min \ E\{e^2[n]\},$$

where $E\{\cdot\}$denotes the expectation operator. In the general case, the coefficients $a_i$may be complex and may be derived for the case where w[n] and s[n] are complex as well. With a complex signal, the matrix to be solved is a Hermitian Toeplitz matrix, rather than symmetric Toeplitz matrix. For simplicity, the following considers only the case where all these quantities are real. The mean square error (MSE) may be rewritten as:

$$\begin{aligned} E\{e^2[n]\} &= E\{(x[n]-s[n])^2\} \\ &= E\{x^2[n]\} + E\{s^2[n]\} - 2E\{x[n]s[n]\} \\ &= E\{\big(\textstyle\sum_{i=0}^{N} a_i w[n-i]\big)^2\} + E\{s^2[n]\} - 2E\{\textstyle\sum_{i=0}^{N} a_i w[n-i]s[n]\}. \end{aligned}$$

To find the vector $[a_0,\ \ldots,\ a_N]$ which minimizes the expression above, calculate its derivative with respect to a_i

$$\begin{aligned}\frac{\partial}{\partial a_i}E\{e^2[n]\} &= 2E\{\left(\textstyle\sum_{j=0}^{N} a_j w[n-j]\right)w[n-i]\} - 2E\{s[n]w[n-i]\} \quad i=0,\ \ldots, \\ &= 2\textstyle\sum_{j=0}^{N} E\{w[n-j]w[n-i]\}a_j - 2E\{w[n-i]s[n]\}.\end{aligned}$$

Assuming that w[n] and s[n] are each stationary and jointly stationary, the sequences $R_w[m]$ and $R_{ws}[m]$ known respectively as the autocorrelation of w[n] and the cross-correlation between w[n] and s[n] can be defined as follows:

$$\begin{aligned}R_w[m] &= E\{w[n]w[n+m]\} \\ R_{ws}[m] &= E\{w[n]s[n+m]\}.\end{aligned}$$

The derivative of the MSE may therefore be rewritten as (notice that $R_{ws}[-i] = R_{sw}[i]$)

$$\frac{\partial}{\partial a_i}E\{e^2[n]\} = 2\sum_{j=0}^{N} R_w[j-i]a_j - 2R_{sw}[i] \quad i=0,\ \ldots,\ N.$$

Letting the derivative be equal to zero results in

$$\sum_{j=0}^{N} R_w[j-i]a_j = R_{sw}[i] \quad i=0,\ \ldots,\ N,$$

which can be rewritten in matrix form

$$\begin{aligned}&\mathbf{Ta} = \mathbf{v} \\ \Rightarrow &\begin{bmatrix} R_w[0] & R_w[1] & \cdots & R_w[N] \\ R_w[1] & R_w[0] & \cdots & R_w[N-1] \\ \vdots & \vdots & \ddots & \vdots \\ R_w[N] & R_w[N-1] & \cdots & R_w[0] \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} R_{sw}[0] \\ R_{sw}[1] \\ \vdots \\ R_{sw}[N] \end{bmatrix}\end{aligned}$$

These equations are known as the Wiener–Hopf equations. The matrix T appearing in the equation is a symmetric Toeplitz matrix. Under suitable conditions on R, these matrices are known to be positive definite and therefore non-singular yielding a unique solution to the determination of the Wiener filter coefficient vector, $\mathbf{a} = \mathbf{T}^{-1}\mathbf{v}$. Furthermore, there exists an efficient algorithm to solve such Wiener–Hopf equations known as the Levinson-Durbin algorithm so an explicit inversion of $\mathbf{T}$ is not required.

Relationship to the least squares filter

The realization of the causal Wiener filter looks a lot like the solution to the least squares estimate, except in the signal processing domain. The least squares solution, for input matrix $\mathbf{X}$ and output vector $\boldsymbol{y}$ is

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^{\mathbf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathbf{T}}\boldsymbol{y}.$$

The FIR Wiener filter is related to the least mean squares filter, but minimizing the error criterion of the latter does not rely on cross-correlations or auto-correlations. Its solution converges to the Wiener filter solution.

Recursive least squares filter

The **Recursive least squares (RLS)** adaptive is an algorithm which recursively finds the filter coefficients that minimize a weighted linear least squares cost function relating to the input signals. This is in contrast to other algorithms such as the least mean squares (LMS) that aim to reduce the mean square error. In the derivation of the RLS, the input signals are considered deterministic, while for the LMS and similar algorithm they are considered stochastic. Compared to most of its competitors, the RLS exhibits extremely fast convergence. However, this benefit comes at the cost of high computational complexity.

Motivation

RLS was discovered by Gauss but lay unused or ignored until 1950 when Plackett rediscovered the original work of Gauss from 1821. In general, the RLS can be used to solve any problem that can be solved by adaptive filters. For example, suppose that a signal d(n) is transmitted over an echoey, noisy channel that causes it to be received as

$$x(n) = \sum_{k=0}^{q} b_n(k)d(n-k) + v(n)$$

where $v(n)$ represents additive noise. We will attempt to recover the desired signal $d(n)$ by use of a $p+1$-tap FIR filter, $\mathbf{w}$:

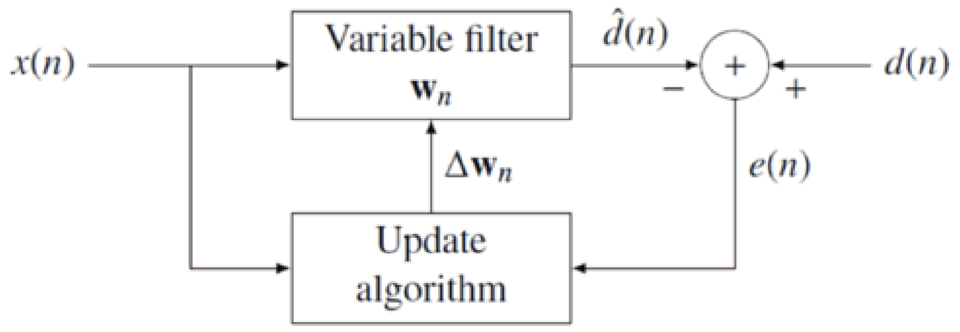
$$\hat{d}(n) = \sum_{k=0}^{p} w_n(k)x(n-k) = \mathbf{w}_n^T\mathbf{x}_n$$

where $\mathbf{x}_n = [x(n) \quad x(n-1) \quad \ldots \quad x(n-p)]^T$ is the vector containing the $p+1$ most recent samples of $x(n)$. Our goal is to estimate the parameters of the filter $\mathbf{w}$, and at each time n we refer to the new least squares estimate by $\mathbf{w_n}$. As time evolves, we would like to avoid completely redoing the least squares algorithm to find the new estimate for $\mathbf{w}_{n+1}$, in terms of $\mathbf{w}_n$.

The benefit of the RLS algorithm is that there is no need to invert matrices, thereby saving computational power. Another advantage is that it provides intuition behind such results as the Kalman filter.

Discussion

The idea behind RLS filters is to minimize a cost function C by appropriately selecting the filter coefficients $\mathbf{w}_n$, updating the filter as new data arrives. The error signal $e(n)$ and desired signal $d(n)$ are defined in the negative feedback diagram below:



The error implicitly depends on the filter coefficients through the estimate $\hat{d}(n)$:

$$e(n) = d(n) - \hat{d}(n)$$

The weighted least squares error function C—the cost function we desire to minimize—being a function of e(n) is therefore also dependent on the filter coefficients:

$$C(\mathbf{w_n}) = \sum_{i=0}^{n} \lambda^{n-i} e^2(i)$$

where $0 < \lambda \leq 1$ is the "forgetting factor" which gives exponentially less weight to older error samples.

The cost function is minimized by taking the partial derivatives for all entries k of the coefficient vector $\mathbf{w}_n$ and setting the results to zero

$$\frac{\partial C(\mathbf{w}_n)}{\partial w_n(k)} = \sum_{i=0}^{n} 2\lambda^{n-i} e(i) \frac{\partial e(i)}{\partial w_n(k)} = -\sum_{i=0}^{n} 2\lambda^{n-i} e(i)\, x(i-k) = 0 \qquad k = 0, 1,$$

Next, replace $e(n)$ with the definition of the error signal

$$\sum_{i=0}^{n} \lambda^{n-i} \left[d(i) - \sum_{l=0}^{p} w_n(l) x(i-l) \right] x(i-k) = 0 \qquad k = 0, 1, \cdots, p$$

Rearranging the equation yields

$$\sum_{l=0}^{p} w_n(l) \left[\sum_{i=0}^{n} \lambda^{n-i} x(i-l) x(i-k) \right] = \sum_{i=0}^{n} \lambda^{n-i} d(i) x(i-k) \qquad k = 0, 1, \cdots, p$$

This form can be expressed in terms of matrices

$$\mathbf{R}_x(n)\, \mathbf{w}_n = \mathbf{r}_{dx}(n)$$

where $\mathbf{R}_x(n)$ is the weighted sample covariance matrix for $x(n)$, and $\mathbf{r}_{dx}(n)$ is the equivalent estimate for the cross-covariance between $d(n)$ and $x(n)$. Based on this expression we find the coefficients which minimize the cost function as

$$\mathbf{w}_n = \mathbf{R}_x^{-1}(n)\, \mathbf{r}_{dx}(n)$$

This is the main result of the discussion.

Choosing λ

The smaller λ is, the smaller contribution of previous samples. This makes the filter more sensitive to recent samples, which means more fluctuations in the filter co-efficients. The $\lambda = 1$ case is referred to as the growing window RLS algorithm. In practice, λ is usually chosen between 0.98 and 1.

Recursive algorithm

The discussion resulted in a single equation to determine a coefficient vector which minimizes the cost function. In this section we want to derive a recursive solution of the form

$$\mathbf{w}_n = \mathbf{w}_{n-1} + \Delta\mathbf{w}_{n-1}$$

where $\Delta\mathbf{w}_{n-1}$ is a correction factor at time $n-1$. We start the derivation of the recursive algorithm by expressing the cross covariance $\mathbf{r}_{dx}(n)$ in terms of $\mathbf{r}_{dx}(n-1)$

$$\begin{aligned}\mathbf{r}_{dx}(n) &= \sum_{i=0}^{n} \lambda^{n-i} d(i)\mathbf{x}(i) \\ &= \sum_{i=0}^{n-1} \lambda^{n-i} d(i)\mathbf{x}(i) + \lambda^0 d(n)\mathbf{x}(n) \\ &= \lambda\mathbf{r}_{dx}(n-1) + d(n)\mathbf{x}(n)\end{aligned}$$

where $\mathbf{x}(i)$ is the $p+1$ dimensional data vector

$$\mathbf{x}(i) = [x(i), x(i-1), \dots, x(i-p)]^T$$

Similarly we express $\mathbf{R}_x(n)$ in terms of $\mathbf{R}_x(n-1)$ by

$$\begin{aligned}\mathbf{R}_x(n) &= \sum_{i=0}^{n} \lambda^{n-i}\mathbf{x}(i)\mathbf{x}^T(i) \\ &= \lambda\mathbf{R}_x(n-1) + \mathbf{x}(n)\mathbf{x}^T(n)\end{aligned}$$

In order to generate the coefficient vector we are interested in the inverse of the deterministic auto-covariance matrix. For that task the Woodbury matrix identity comes in handy. With

$A = \lambda\mathbf{R}_x(n-1)$ is $(p+1)$-by-$(p+1)$

$U = \mathbf{x}(n)$ is $(p+1)$-by-1

$V = \mathbf{x}^T(n)$ is 1-by-$(p+1)$

$C = \mathbf{I}_1$ is the 1-by-1 identity matrix

The Woodbury matrix identity follows

$$\begin{aligned}\mathbf{R}_x^{-1}(n) - [\lambda\mathbf{R}_x(n-1) + \mathbf{x}(n)\mathbf{x}^T(n)]^{-1} \\ &= \lambda^{-1}\mathbf{R}_x^{-1}(n-1) \\ &\quad -\lambda^{-1}\mathbf{R}_x^{-1}(n-1)\mathbf{x}(n) \\ &\quad \{1 + \mathbf{x}^T(n)\lambda^{-1}\mathbf{R}_x^{-1}(n-1)\mathbf{x}(n)\}^{-1}\mathbf{x}^T(n)\lambda^{-1}\mathbf{R}_x^{-1}(n-1)\end{aligned}$$

To come in line with the standard literature, we define

$$\begin{aligned}\mathbf{P}(n) &= \mathbf{R}_x^{-1}(n) \\ &= \lambda^{-1}\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)\end{aligned}$$

where the gain vector $g(n)$is

$$\begin{aligned}\mathbf{g}(n) &= \lambda^{-1}\mathbf{P}(n-1)\mathbf{x}(n)\{1 + \mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)\mathbf{x}(n)\}^{-1} \\ &= \mathbf{P}(n-1)\mathbf{x}(n)\{\lambda + \mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{x}(n)\}^{-1}\end{aligned}$$

Before we move on, it is necessary to bring $\mathbf{g}(n)$into another form

$$\mathbf{g}(n)\{1 + \mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)\mathbf{x}(n)\} = \lambda^{-1}\mathbf{P}(n-1)\mathbf{x}(n)$$

$$\mathbf{g}(n) + \mathbf{g}(n)\mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)\mathbf{x}(n) = \lambda^{-1}\mathbf{P}(n-1)\mathbf{x}(n)$$

Subtracting the second term on the left side yields

$$\begin{aligned}\mathbf{g}(n) &= \lambda^{-1}\mathbf{P}(n-1)\mathbf{x}(n) - \mathbf{g}(n)\mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)\mathbf{x}(n) \\ &= \lambda^{-1}[\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{x}^T(n)\mathbf{P}(n-1)]\mathbf{x}(n)\end{aligned}$$

With the recursive definition of $\mathbf{P}(n)$the desired form follows

$$\mathbf{g}(n) = \mathbf{P}(n)\mathbf{x}(n)$$

Now we are ready to complete the recursion. As discussed

$$\begin{aligned}\mathbf{w}_n &= \mathbf{P}(n)\,\mathbf{r}_{dx}(n)\\ &= \lambda\mathbf{P}(n)\,\mathbf{r}_{dx}(n-1) + d(n)\mathbf{P}(n)\mathbf{x}(n)\end{aligned}$$

The second step follows from the recursive definition of $\mathbf{r}_{dx}(n)$. Next we incorporate the recursive definition of $\mathbf{P}(n)$together with the alternate form of $\mathbf{g}(n)$and get

$$\begin{aligned}\mathbf{w}_n &= \lambda\left[\lambda^{-1}\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)\right]\mathbf{r}_{dx}(n-1) + d(n)\mathbf{g}(\\ &= \mathbf{P}(n-1)\mathbf{r}_{dx}(n-1) - \mathbf{g}(n)\mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{r}_{dx}(n-1) + d(n)\mathbf{g}(\\ &= \mathbf{P}(n-1)\mathbf{r}_{dx}(n-1) + \mathbf{g}(n)\left[d(n) - \mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{r}_{dx}(n-1)\right]\end{aligned}$$

With $\mathbf{w}_{n-1} = \mathbf{P}(n-1)\mathbf{r}_{dx}(n-1)$we arrive at the update equation

$$\begin{aligned}\mathbf{w}_n &= \mathbf{w}_{n-1} + \mathbf{g}(n)\left[d(n) - \mathbf{x}^T(n)\mathbf{w}_{n-1}\right]\\ &= \mathbf{w}_{n-1} + \mathbf{g}(n)\alpha(n)\end{aligned}$$

where $\alpha(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}_{n-1}$is the a priori error. Compare this with the a posteriori error; the error calculated after the filter is updated:

$$e(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}_n$$

That means we found the correction factor

$$\Delta\mathbf{w}_{n-1} = \mathbf{g}(n)\alpha(n)$$

This intuitively satisfying result indicates that the correction factor is directly proportional to both the error and the gain vector, which controls how much sensitivity is desired, through the weighting factor, λ.

RLS algorithm summary

The RLS algorithm for a p-th order RLS filter can be summarized as

Parameters: $p =$filter order

$\lambda =$forgetting factor

$\delta =$value to initialize $\mathbf{P}(0)$

Initialization: $\mathbf{w}(n) = 0$,

$x(k) = 0, k = -p, \ldots, -1$,

$d(k) = 0, k = -p, \ldots, -1$

$\mathbf{P}(0) = \delta^{-1} I$ where I is the identity matrix of rank $p + 1$

Computation: For $n = 1, 2, \ldots$

$$\mathbf{x}(n) = \begin{bmatrix} x(n) \\ x(n-1) \\ \vdots \\ x(n-p) \end{bmatrix}$$

$$\alpha(n) = d(n) - \mathbf{x}^T(n)\mathbf{w}(n-1)$$

$$\mathbf{g}(n) = \mathbf{P}(n-1)\mathbf{x}^*(n)\left\{\lambda + \mathbf{x}^T(n)\mathbf{P}(n-1)\mathbf{x}^*(n)\right\}^{-1}$$

$$\mathbf{P}(n) = \lambda^{-1}\mathbf{P}(n-1) - \mathbf{g}(n)\mathbf{x}^T(n)\lambda^{-1}\mathbf{P}(n-1)$$

$$\mathbf{w}(n) = \mathbf{w}(n-1) + \alpha(n)\mathbf{g}(n).$$

Note that the recursion for P follows an Algebraic Riccati equation and thus draws parallels to the Kalman filter.

Lattice recursive least squares filter (LRLS)

The **Lattice Recursive Least Squares** adaptive filter is related to the standard RLS except that it requires fewer arithmetic operations (order N). It offers additional advantages over conventional LMS algorithms such as faster convergence rates, modular structure, and insensitivity to variations in eigenvalue spread of the input correlation matrix. The LRLS algorithm described is based on a posteriori errors and includes the normalized form. The derivation is similar to the standard RLS algorithm and is based on the definition of $d(k)$. In the forward prediction case, we have $d(k) = x(k)$with the input signal $x(k-1)$as the most up to date sample. The backward prediction case is $d(k) = x(k-i-1)$, where i is the index of the sample in the past we want to predict, and the input signal $x(k)$is the most recent sample.

Parameter Summary

$\kappa_f(k,i)$is the forward reflection coefficient

$\kappa_b(k,i)$is the backward reflection coefficient

$e_f(k,i)$represents the instantaneous a posteriori forward prediction error

$e_b(k,i)$represents the instantaneous a posteriori backward prediction error

$\xi^d_{b_{min}}(k,i)$is the minimum least-squares backward prediction error

$\xi^d_{f_{min}}(k,i)$is the minimum least-squares forward prediction error

$\gamma(k,i)$is a conversion factor between a priori and a posteriori errors

$v_i(k)$are the feedforward multiplier coefficients.

$\epsilon$is a small positive constant that can be 0.01

LRLS Algorithm Summary

The algorithm for a LRLS filter can be summarized as

Initialization:

For i = 0,1,...,N

$\delta(-1,i) = \delta_D(-1,i) = 0$ (if x(k) = 0 for k < 0)

$\xi^d_{b_{min}}(-1,i) = \xi^d_{f_{min}}(-1,i) = \epsilon$

$\gamma(-1,i) = 1$

```
        $e_b(-1,i) = 0$
        End
Computation:
        For k ≥ 0
        $\gamma(k,0) = 1$
        $e_b(k,0) = e_f(k,0) = x(k)$
        $\xi^d_{b_{min}}(k,0) = \xi^d_{f_{min}}(k,0) = x^2(k) + \lambda\xi^d_{f_{min}}(k-1,0)$
        $e(k,0) = d(k)$
        For i = 0,1,...,N
        $\delta(k,i) = \lambda\delta(k-1,i) + \frac{e_b(k-1,i)e_f(k,i)}{\gamma(k-1,i)}$
        $\gamma(k,i+1) = \gamma(k,i) - \frac{e_b^2(k,i)}{\xi^d_{b_{min}}(k,i)}$
        $\kappa_b(k,i) = \frac{\delta(k,i)}{\xi^d_{f_{min}}(k,i)}$
        $\kappa_f(k,i) = \frac{\delta(k,i)}{\xi^d_{b_{min}}(k-1,i)}$
        $e_b(k,i+1) = e_b(k-1,i) - \kappa_b(k,i)e_f(k,i)$
        $e_f(k,i+1) = e_f(k,i) - \kappa_f(k,i)e_b(k-1,i)$
        $\xi^d_{b_{min}}(k,i+1) = \xi^d_{b_{min}}(k-1,i) - \delta(k,i)\kappa_b(k,i)$
        $\xi^d_{f_{min}}(k,i+1) = \xi^d_{f_{min}}(k,i) - \delta(k,i)\kappa_f(k,i)$
        Feedforward Filtering
        $\delta_D(k,i) = \lambda\delta_D(k-1,i) + \frac{e(k,i)e_b(k,i)}{\gamma(k,i)}$
        $v_i(k) = \frac{\delta_D(k,i)}{\xi^d_{b_{min}}(k,i)}$
        $e(k,i+1) = e(k,i) - v_i(k)e_b(k,i)$
        End
    End
```

Normalized lattice recursive least squares filter (NLRLS)

The normalized form of the LRLS has fewer recursions and variables. It can be calculated by applying a normalization to the internal variables of the algorithm which will keep their magnitude bounded by one. This is generally not used in real-time applications because of the number of division and square-root operations which comes with a high computational load.

NLRLS algorithm summary

The algorithm for a NLRLS filter can be summarized as

```
Initializatio
n:
        For i = 0,1,...,N
        $\overline{\delta}(-1,i) = 0$ (if x(k) = d(k) = 0 for k < 0)
        $\overline{\delta}_D(-1,i) = 0$
        $\overline{e}_b(-1,i) = 0$
```

End

$$\sigma_x^2(-1) = \lambda\sigma_d^2(-1) = \epsilon$$

Computation:

For k ≥ 0

$\sigma_x^2(k) = \lambda\sigma_x^2(k-1) + x^2(k)$ (Input signal energy)

$\sigma_d^2(k) = \lambda\sigma_d^2(k-1) + d^2(k)$ (Reference signal energy)

$$\overline{e}_b(k,0) = \overline{e}_f(k,0) = \frac{x(k)}{\sigma_x(k)}$$

$$\overline{e}(k,0) = \frac{d(k)}{\sigma_d(k)}$$

For i = 0,1,...,N

$$\overline{\delta}(k,i) = \delta(k-1,i)\sqrt{(1-\overline{e}_b^2(k-1,i))(1-\overline{e}_f^2(k,i))} + \overline{e}_b(k-1,i)$$

$$\overline{e}_b(k,i+1) = \frac{\overline{e}_b(k-1,i) - \delta(k,i)\overline{e}_f(k,i)}{\sqrt{(1-\overline{\delta}^2(k,i))(1-\overline{e}_f^2(k,i))}}$$

$$\overline{e}_f(k,i+1) = \frac{\overline{e}_f(k,i) - \delta(k,i)\overline{e}_b(k-1,i)}{\sqrt{(1-\overline{\delta}^2(k,i))(1-\overline{e}_b^2(k-1,i))}}$$

Feedforward Filter

$$\overline{\delta}_D(k,i) = \overline{\delta}_D(k-1,i)\sqrt{(1-\overline{e}_b^2(k,i))(1-\overline{e}^2(k,i))} + \overline{e}(k,i)\overline{e}_b(k,i$$

```
$\overline{e}(k,i+1) = \frac{1}{\sqrt{(1-\overline{e}_b^2(k,i))(1-\overline{\delta}_D^2(k,i))}}[\overline{e}(k,i) - \overline{\delta}_D(k,i)\overline{e}_b(k,i$
 End
End
```