Concise DSP Tutorial

Home

ROBERT W. STEWART DANIEL GARCÍA-ALÍS

Digital Communications

BERNARD SKLAR 2nd Edition

the *Companion* **CD**

- This document contains a short tutorial on fundamental DSP algorithms and applications.
- Specifically we review frequency domain techniques, digital filters, and adaptive DSP.
- Open the Acrobat bookmarks on the left to navigate the various sections.

- *Simulations powered by*

SystemView BY ELANIX®

- *Tutorial by*

DSP*edia*® www.dspedia.com

1 Introduction

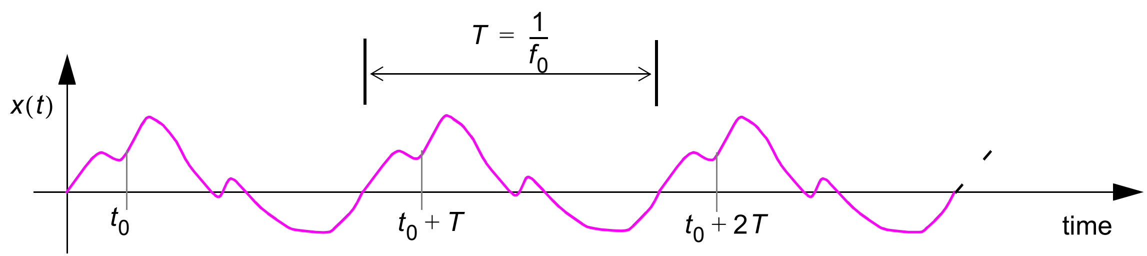
In this document we review some standard mathematics and key analysis points of digital signal processing. Specifically this review will cover frequency domain analysis, digital filtering and adaptive digital filtering. The aim of the this document is to provide additional mathematical information to the *Getting Started* and *Book Exercise* books on the companion CD.

2 Frequency Domain Analysis

2.1 Fourier Series

There exists mathematical theory called the Fourier series that allows any periodic waveform in time to be decomposed into a sum of harmonically related sine and cosine waveforms. The first requirement in realising the Fourier series is to calculate the fundamental period, T, which is the shortest time over which the signal repeats, i.e. for a signal $x(t)$, then:

$$x(t) = x(t+T) = x(t+2T) = \ldots = x(t+kT) \quad (1)$$



The (fundamental) period of a signal $x(t)$ identified as T. The fundamental frequency, f_0, is calculated as $f_0 = 1/T$. Clearly $x(t_0) = x(t_0 + T) = x(t_0 + 2T)$.

For a periodic signal with fundamental period T seconds, the *Fourier series* represents this signal as a sum of sine and cosine components that are harmonics of the fundamental frequency, $f_0 = 1/T$ Hz. The Fourier series can be written in a number of different ways:

$$
\begin{aligned}
x(t) &= \sum_{n=0}^{\infty} A_n \cos\left(\frac{2\pi nt}{T}\right) + \sum_{n=1}^{\infty} B_n \sin\left(\frac{2\pi nt}{T}\right) \\
&= A_0 + \sum_{n=1}^{\infty} \left[A_n \cos\left(\frac{2\pi nt}{T}\right) + B_n \sin\left(\frac{2\pi nt}{T}\right)\right] \\
&= A_0 + \sum_{n=1}^{\infty} [A_n \cos(2\pi n f_0 t) + B_n \sin(2\pi n f_0 t)] \\
&= A_0 + \sum_{n=1}^{\infty} [A_n \cos(n\omega_0 t) + B_n \sin(n\omega_0 t)] \\
&= \sum_{n=0}^{\infty} [A_n \cos(n\omega_0 t) + B_n \sin(n\omega_0 t)] \\
&= A_0 + A_1 \cos(\omega_0 t) + A_2 \cos(2\omega_0 t) + A_2 \cos(3\omega_0 t) + \ldots \\
&\quad + B_1 \sin(\omega_0 t) + B_2 \sin(2\omega_0 t) + B_2 \sin(3\omega_0 t) + \ldots
\end{aligned}
\tag{2}
$$

where A_n and B_n are the amplitudes of the various cosine and sine waveforms, and angular frequency is denoted by $\omega_0 = 2\pi f_0$ radians/second.

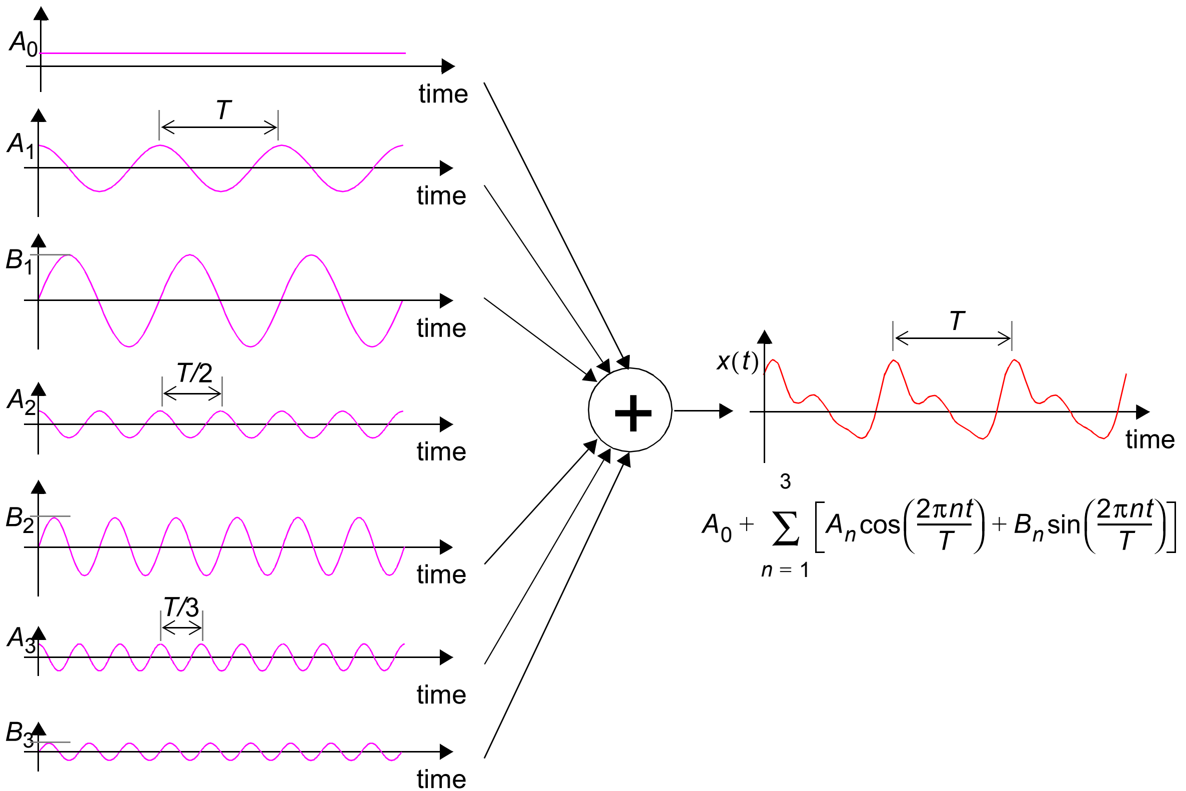
Depending on the actual problem being solved we can choose to specify the fundamental periodicity of the waveform in terms of the period (T), frequency (f_0), or angular frequency (ω_0) as shown in Eq. 2. Note that there is actually no requirement to specifically include a B_0 term since $\sin 0 = 0$, although there is an A_0 term, since $\cos 0 = 1$, which represents any DC component that may be present in the signal.

In more descriptive language the above Fourier series says that any periodic signal can be reproduced by adding a (possibly infinite) series of harmonically related sinusoidal waveforms of amplitudes A_n or B_n. Therefore, if a periodic signal with a fundamental period of say 0.01 seconds is identified, then the Fourier series will allow this waveform to be represented as a sum of various cosine and sine waves at frequencies of 100 Hz (the fundamental frequency, f_0), 200Hz, 300Hz (the harmonic frequencies $2f_0$, $3f_0$) and so on. The amplitudes of these cosine and sine waves are given by A_0, A_1, B_1, A_2, B_2, A_3 ..... and so on.

So how are the values of A_n and B_n calculated? The answer can be derived by some basic trigonometry. Taking the last line of Eq. 2, if we multiply both sides by $\cos(p\omega_0 t)$, where p is an arbitrary positive integer, then we get:

$$\cos(p\omega_0 t)x(t) = \cos(p\omega_0 t)\sum_{n=0}^{\infty}[A_n\cos(n\omega_0 t) + B_n\sin(n\omega_0 t)] \quad (3)$$

Now we take the average of one fundamental period of both sides, this can be done by integrating the functions over any one period T:



$$A_0 + \sum_{n=1}^{3} \left[A_n \cos\left(\frac{2\pi n t}{T}\right) + B_n \sin\left(\frac{2\pi n t}{T}\right) \right]$$

Fourier series for a periodic signal $x(t)$. If we analyse a periodic signal and realizethe cosine and sine wave Fourier coefficients of appropriate amplitudes A_n and B_n, then summing these components will lead to exactly the original signal.

$$\int_0^T \cos(p\omega_0 t)x(t)dt = \int_0^T \left\{ \cos(p\omega_0 t) \sum_{n=0}^{T} A_n \cos(n\omega_0 t) + \sum_{n=0}^{T} B_n \sin(n\omega_0 t) \right\} dt$$

$$= \sum_{n=0}^{\infty} \int_0^T \{A_n \cos(p\omega_0 t)\cos(n\omega_0 t)\}dt + \sum_{n=0}^{\infty} \int_0^T \{B_n \cos(p\omega_0 t)\sin(n\omega_0 t)\}dt \qquad (4)$$

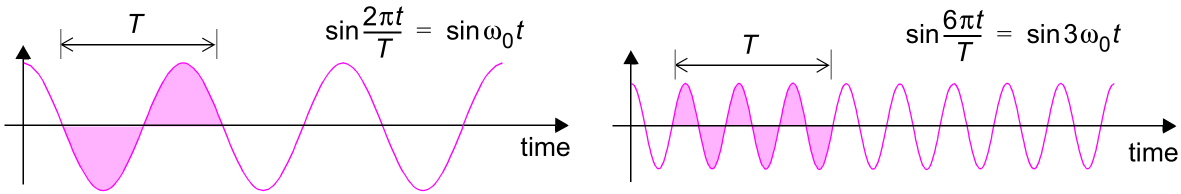
Noting the zero value of the second term in the last line of Eq. 4, i.e.:

$$\int_0^T \{B_n \cos(p\omega_0 t)\sin(n\omega_0 t)\}dt = \frac{B_n}{2}\int_0^T (\sin(p+n)\omega_0 t - \sin(p-n)\omega_0 t)dt$$

$$= \frac{B_n}{2}\int_0^T \sin\left[\frac{(p+n)2\pi t}{T}\right]dt - \frac{B_n}{2}\int_0^T \sin\left[\frac{(p-n)2\pi t}{T}\right]dt \qquad (5)$$

$$= 0$$

using the trigonometric identity $2\cos A \sin B = \sin(A+B) - \sin(A-B)$ and noting that the integral over one period, T, of any harmonic of the term $\sin[2\pi t/T]$ is zero:



The integral over T of any sine/cosine waveform of frequency $f_0 = 1/T$ or harmonics thereof, $2f_0, 2f_0, 3f_0, \ldots$ is zero, regardless of the amplitude or phase of the signal.

Eq. 5 is true for all values of the positive integers p and n.

For the first term in the last line of Eq. 4 the average is only zero if $p \neq n$, i.e.:

$$\int_0^T A_n \cos(p\omega_0 t)\cos(n\omega_0 t)\; dt = \frac{A_n}{2}\int_0^T (\cos(p+n)\omega_0 t + \cos(p-n)\omega_0 t)\,dt = 0, \quad p \neq n \tag{6}$$

this time using the trigonometric identity $2\cos A\cos B = \cos(A+B) + \cos(A-B)$.

If $p = n$ then:

$$\begin{aligned}\int_0^T A_n\cos(n\omega_0 t)\cos(n\omega_0 t)\;dt &= A_n\int_0^T\cos^2(n\omega_0 t)\;dt \\ &= \frac{A_n}{2}\int_0^T(1+\cos 2n\omega_0 t)dt = \frac{A_n}{2}\int_0^T 1\,dt = \frac{A_n t}{2}\bigg|_0^T = \frac{A_n T}{2}\end{aligned} \tag{7}$$

Therefore using Eqs. 5, 6, 7 in Eq. 4 we note that:

$$\int_0^T \cos(p\omega_0 t)x(t)dt = \frac{A_n T}{2} \tag{8}$$

and therefore, since $p = n$:

$$A_n = \frac{2}{T}\int_0^T x(t)\cos(n\omega_0 t)dt \tag{9}$$

By premultiplying and time averaging Eq. 4 by $\sin(p\omega_0 t)$ and using a similar set of simplifications to Eqs. 5, 6, 7 we can similarly show that:

$$B_n = \frac{2}{T}\int_0^T x(t)\sin(n\omega_0 t)dt \tag{10}$$

Hence the three key equations for calculating the Fourier series of a periodic signal with fundamental period T are:

$$
\begin{aligned}
x(t) &= \sum_{n=0}^{\infty} A_n \cos\left(\frac{2\pi nt}{T}\right) + \sum_{n=1}^{\infty} B_n \sin\left(\frac{2\pi nt}{T}\right) \\
A_n &= \frac{2}{T}\int_0^T x(t)\cos(n\omega_0 t)\,dt \\
B_n &= \frac{2}{T}\int_0^T x(t)\sin(n\omega_0 t)\,dt
\end{aligned}
\tag{11}
$$

Fourier Series Equations

2.2 Amplitude/Phase Representation

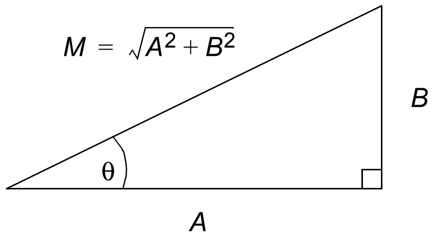
It is often useful to abbreviate the notation of the Fourier series such that the series is a sum of cosine (or sine) only terms with a phase shift. To perform this notational simplification, first consider the simple trigonometric function:

$$A\cos\omega t + B\sin\omega t \tag{12}$$

where A and B are real numbers. If we introduce another variable, M such that $M = \sqrt{A^2 + B^2}$ then:

$$
\begin{aligned}
A\cos\omega t + B\sin\omega t &= \frac{\sqrt{A^2+B^2}}{\sqrt{A^2+B^2}}(A\cos\omega t + B\sin\omega t) \\
&= M\left(\frac{A}{\sqrt{A^2+B^2}}\cos\omega t + \frac{B}{\sqrt{A^2+B^2}}\sin\omega t\right) \\
&= M(\cos\theta\cos\omega t + \sin\theta\sin\omega t) \\
&= M\cos(\omega t - \theta) \\
&= \sqrt{A^2+B^2}\cos(\omega t - \{\tan^{-1}B/A\})
\end{aligned}
\tag{13}
$$

since θ is the angle made by a right angle triangle of hypotenuse M and sides of A and B, i.e. $\tan^{-1}(B/A) = \theta$.



A simple right angled triangle with arbitrary length sides of A and B. The sine of the angle θ is the ratio of the opposite side over the hypotenuse, B/M and the cosine of the angle θ is the ratio of the adjacent side over the hypotenuse, A/M. The tangent of the angle θ is the ratio of the opposite side over the adjacent side, B/A.

This result shows that the sum of a sine and a cosine waveform of arbitrary amplitudes is a sinusoidal signal of the same frequency but different amplitude and phase from the original sine and cosine terms. Using this result of Eq. 13 to combine each sine and cosine term, we can rewrite the Fourier series of Eq. 2 as:

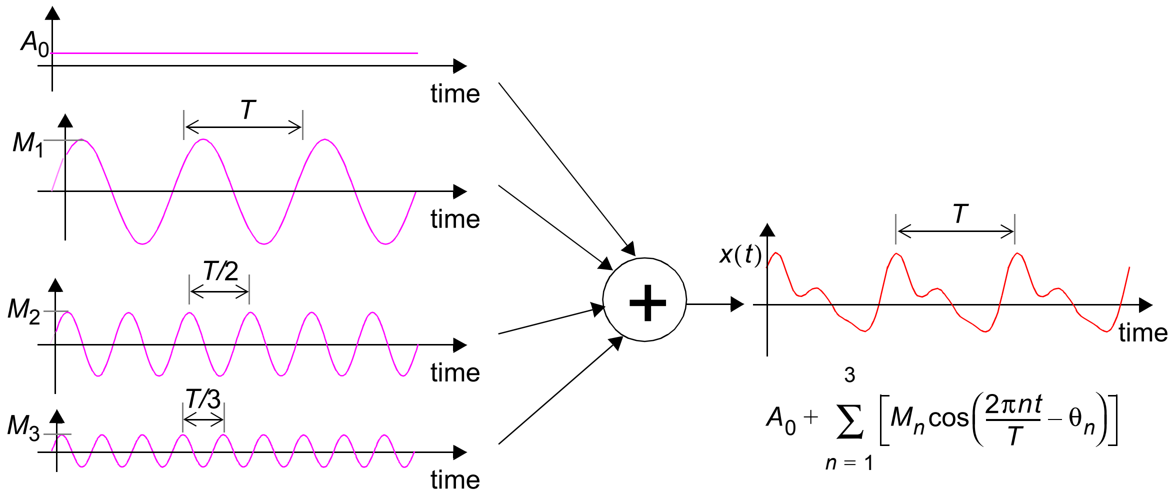
$$x(t) = \sum_{n=0}^{\infty} A_n \cos\left(\frac{2\pi nt}{T}\right) + \sum_{n=1}^{\infty} B_n \sin\left(\frac{2\pi nt}{T}\right)$$

$$x(t) = \sum_{n=0}^{\infty} M_n \cos(n\omega_0 t - \theta_n) \tag{14}$$

$$\theta_n = \tan^{-1} B_n / A_n$$

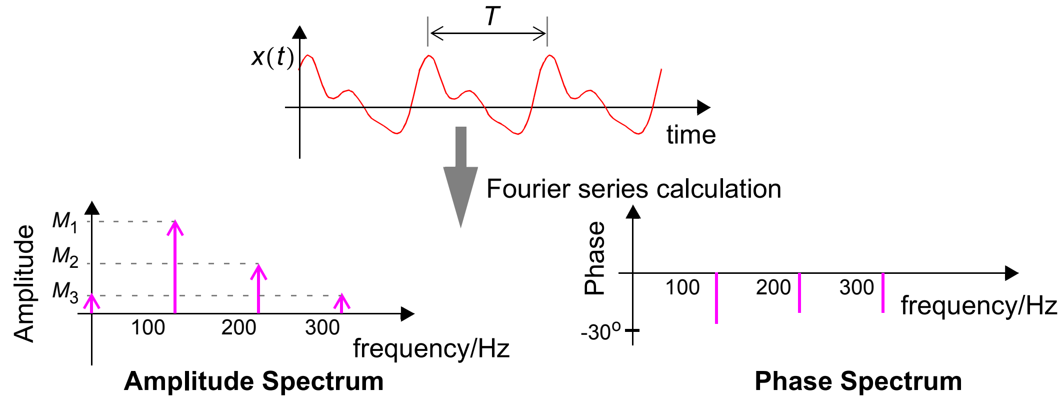
$$M_n = \sqrt{A_n^2 + B_n^2}$$

where A_n and B_n are calculated as before using Eqs. 9 and 10.



Comparing this Fourier series with the one on page 6 note that the sine and cosine terms have been combined for each frequency to produce a single cosine waveform of amplitude $M_n = \sqrt{A_n^2 + B_n^2}$ and phase $\theta_n = B/A$.

From this representation of the Fourier series, we can plot an *amplitude line spectrum* and a *phase spectrum*, the former shows the amplitudes of the different sinusoids and the latter their phases:



The Fourier series components of the form: $M_n \cos(2\pi f_0 t - \theta_n)$. The amplitude spectrum shows the amplitudes of each of the sine waves, and the phase spectrum shows the phase shift (in degrees in this example) of each cosine component. Note that the combination of the amplitude and phase spectrum completely defines the time signal.

2.3 Complex Exponential Representation

It can be useful and instructive to represent the *Fourier series* in terms of complex exponentials rather than sine and cosine waveforms (in the derivation presented below we will assume that the signal under analysis is real valued, although the result extends easily to complex signals). From Euler's formula, note that:

$$e^{j\omega} = \cos\omega + j\sin\omega\cos\omega \Rightarrow \cos\omega = \frac{e^{j\omega}+e^{-j\omega}}{2} \quad \text{and} \quad \sin\omega = \frac{e^{j\omega}-e^{-j\omega}}{2j} \qquad (15)$$

Substituting the complex exponential definitions for sine and cosine in Eq. 2 and rearranging gives:

$$\begin{aligned} x(t) &= A_0 + \sum_{n=1}^{\infty} A_n\cos(n\omega_0 t) + B_n\sin(n\omega_0 t) \\ &= A_0 + \sum_{n=1}^{\infty}\left[A_n\left(\frac{e^{jn\omega_0 t}+e^{-jn\omega_0 t}}{2}\right) + B_n\left(\frac{e^{jn\omega_0 t}-e^{-jn\omega_0 t}}{2j}\right)\right] \\ &= A_0 + \sum_{n=1}^{\infty}\left[\left(\frac{A_n}{2}+\frac{B_n}{2j}\right)e^{jn\omega_0 t} + \left(\frac{A_n}{2}-\frac{B_n}{2j}\right)e^{-jn\omega_0 t}\right] \\ &= A_0 + \sum_{n=1}^{\infty}\left(\frac{A_n - jB_n}{2}\right)e^{jn\omega_0 t} + \sum_{n=1}^{\infty}\left(\frac{A_n + jB_n}{2}\right)e^{-jn\omega_0 t} \end{aligned} \qquad (16)$$

For the second summation term, if the sign of the complex sinusoid is negated and the summation limits are reversed, then we can rewrite as:

$$\begin{aligned} x(t) &= A_0 + \sum_{n=1}^{\infty}\left(\frac{A_n - jB_n}{2}\right)e^{jn\omega_0 t} + \sum_{n=-\infty}^{-1}\left(\frac{A_n + jB_n}{2}\right)e^{jn\omega_0 t} \\ &= \sum_{n=-\infty}^{\infty} C_n e^{jn\omega_0 t} \end{aligned} \tag{17}$$

Writing C_n in terms of the Fourier series coefficients of Eqs. 9 and 10 gives:

$$\begin{aligned} C_0 &= A_0 \\ C_n &= (A_n - jB_n)/2 \quad \text{for} \quad n > 0 \\ C_n &= (A_n + jB_n)/2 \quad \text{for} \quad n < 0 \end{aligned} \tag{18}$$

From Eq 18, note that for $n \geq 0$:

$$C_n = \frac{A_n - jB_n}{2} = \frac{1}{T}\int_0^T x(t)\cos(n\omega_0 t)dt - j\frac{1}{T}\int_0^T x(t)\sin(n\omega_0 t)dt$$

$$= \frac{1}{T}\int_0^T x(t)[\cos(n\omega_0 t) - j\sin(n\omega_0 t)]dt \tag{19}$$

$$= \frac{1}{T}\int_0^T x(t)e^{-jn\omega_o t}dt$$

For $n < 0$ it is clear from Eq. 18 that $C_n = C^*_{-n}$ where "*" denotes complex conjugate. Therefore we have now defined the Fourier series of a real valued signal using a complex analysis and a synthesis equation:

$$x(t) = \sum_{n = \infty}^{\infty} C_n e^{jn\omega_0 t} \qquad \text{Synthesis}$$

$$C_n = \frac{1}{T}\int_0^T x(t)e^{-jn\omega_o t}dt \qquad \text{Analysis} \tag{20}$$

Complex Fourier Series Equations

The complex Fourier series also introduces the concept of so called "negative frequecies" whereby we view signals of the form $e^{j2\pi f_0}$ as a positive complex sinusoid of frequency f_0 Hz, and signals of the form $e^{-j2\pi f_0}$ as a complex sinusoid of frequency $-f_0$ Hz.

Note that the complex Fourier series is more notationally compact, and probably simpler to work with than the general Fourier series (the "probably" depends on how clear you are in dealing with complex exponentials!). Also, if the signal being analysed is in fact complex, the general Fourier series of Eq. 2 (see *Fourier Series*) is insufficient but Eqs. 20 can be used. (For complex signals the coefficient relationship in Eq. 18 will not in general hold).

Assuming the waveform being analysed is real (usually the case), then it is easy to convert C_n coefficients into A_n and B_n. Also note from Eq. 14 and Eq. 18 that:

$$M_n = \sqrt{A_n^2 + B_n^2} = 2|C_n| \tag{21}$$

noting that $|C_n| = \sqrt{A_n^2 + B_n^2}/2$. Clearly we can also note that for the complex number C_n:

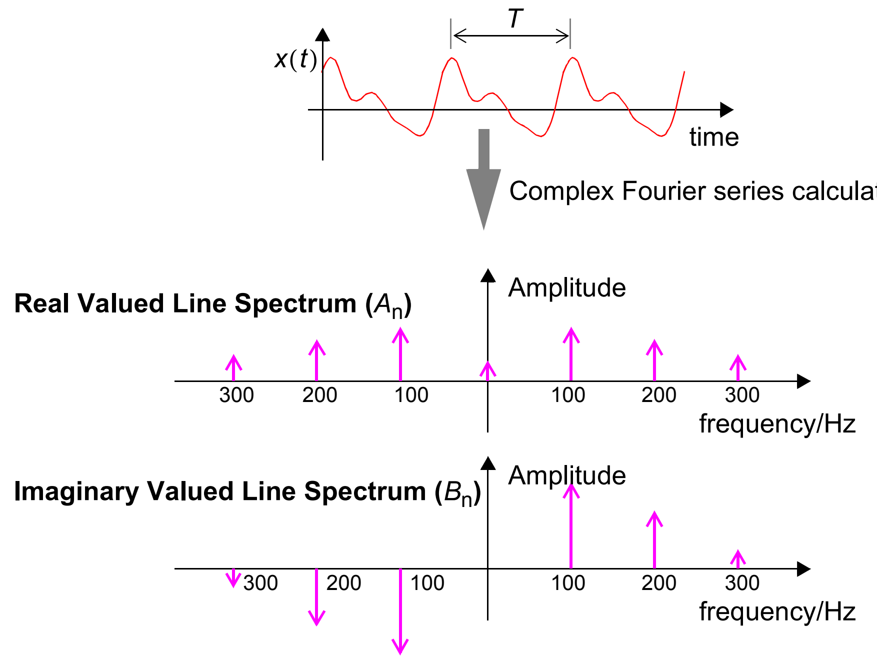
$$\angle C_n = \tan^{-1}\frac{B}{A} = \theta_n \qquad \text{i.e. } C_n = |C_n| e^{j\theta_n} \tag{22}$$

Therefore although a complex exponential does not as such exist as a real world (single wire voltage) signal, we can easily convert from a complex exponential to a real world sinusoid simply by taking the real or imaginary part of the complex Fourier coefficients and use in the Fourier series equation (see Eq. 2):

$$x(t) = \sum_{n=0}^{\infty} [A_n \cos(n\omega_0 t) + B_n \sin(n\omega_0 t)] \tag{23}$$

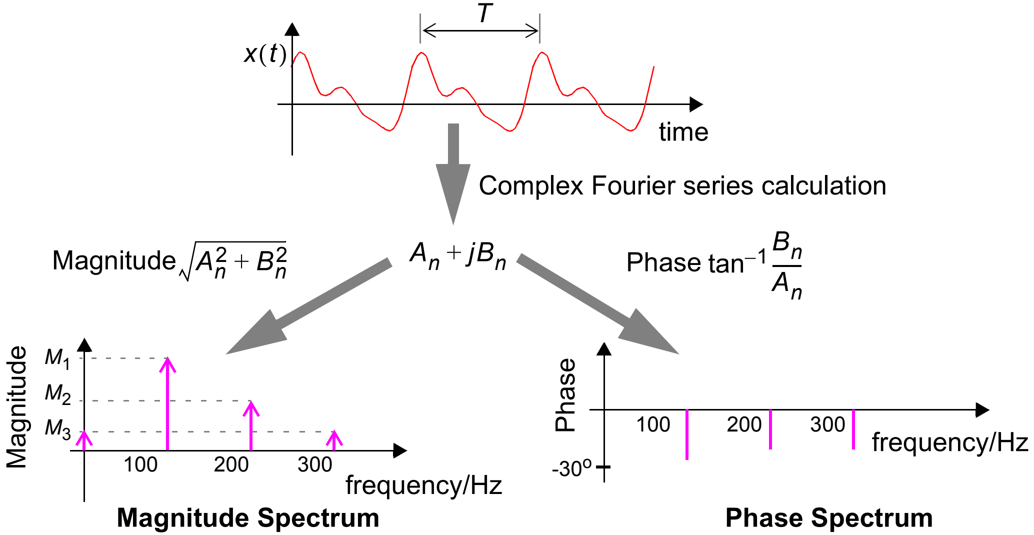
There are of course certain time domain signals which can be considered as being complex, i.e. having a separate real and imaginary components. This type of signal can be found in some digital communication systems or may be created within a DSP system to allow certain types of computation to be performed.

If a signal is decomposed into its complex Fourier series, the resulting values for the various components can be plotted as a line spectrum. As we now have both complex and real values and positive and negative frequencies, this will require two plots, one for the imaginary components and one for the real components:



The complex Fourier series line spectra. Note that there are both positive and negative frequencies, and for the complex Fourier series of a real valued signal the real line spectrum is symmetrical about $f = 0$ and the imaginary spectrum has point symmetry about the origin.

Rather than showing the real and imaginary line spectra, it is more usual to plot the magnitude spectrum and phase spectrum:



Calculating the magnitude and phase spectra from the complex Fourier series. For a real valued signal, the result will be identical, except for a magnitude scaling factor of 2, to that obtained from the amplitude phase form of the Fourier series as on page 14. As both spectra are symmetrical about the y-axis the negative frequency values are not plotted.

The "ease" of working with complex exponentials over sines and cosines can be illustrated by asking the reader to simplify the following equation to a sum of sine waves:

$$\sin(\omega_1 t)\sin(\omega_2 t) \qquad (24)$$

This requires the recollection (or re-derivation!) of trigonometric identities to yield:

$$\sin(\omega_1 t)\sin(\omega_2 t) = \frac{1}{2}\cos(\omega_1 - \omega_2)t + \frac{1}{2}\cos(\omega_1 + \omega_2)t \qquad (25)$$

While not particularly arduous, it is somewhat easier to simplify the following expression to a sum of complex exponentials:

$$e^{j\omega_1 t}e^{j\omega_2 t} = e^{j(\omega_1 + \omega_2)t} \qquad (26)$$

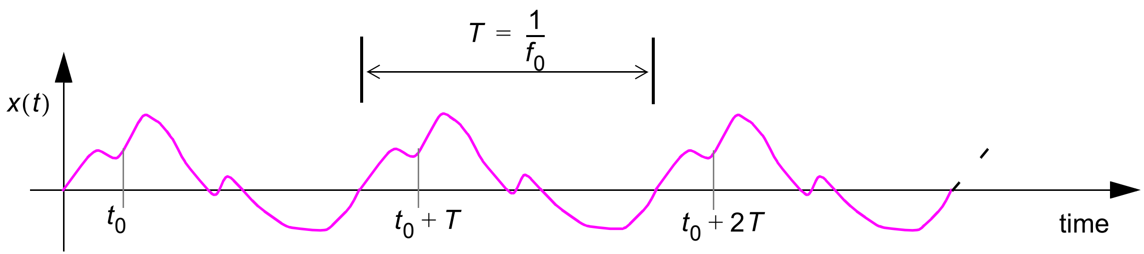
Although a seemingly simple comment, this is the basis of using complex exponentials rather than sines and cosines; they make the maths easier. Of course, in situations where the signal being analysed is complex (or *analytic*), then the complex exponential Fourier series *must* be used!

2.4 Fourier Transform

The Fourier series (rather than transform) allows a periodic signal to be broken down into a sum of real valued sine and cosine waves (in the case of a real valued signal) or more generally a sum of complex exponentials. However most signals are aperiodic, i.e. not periodic. Therefore the Fourier transform was derived in order to analyse the frequency content of an aperiodic signal.

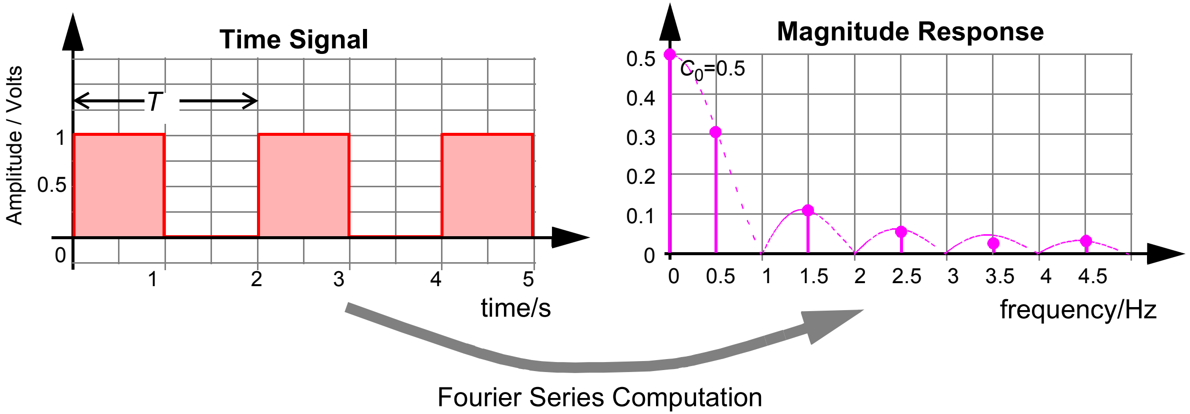
Consider the complex Fourier series of a periodic signal:

$$x(t) = \sum_{n=-\infty}^{\infty} C_n e^{jn\omega_0 t}$$
$$C_n = \frac{1}{T}\int_0^T x(t) e^{-jn\omega_0 t} dt \qquad (27)$$



A periodic signal $x(t)$ with period T. The fundamental frequency, f_0 is calculated simply as $f_0 = 1/T$. Clearly $x(t_0) = x(t_0 + T) = x(t_0 + 2T)$.

The period of the signal has been identified as T and the fundamental frequency is $f_0 = 1/T$. Therefore the Fourier series harmonics occur at frequencies $f_0, 2f_0, 3f_0, \ldots$.



Magnitude response of a (periodic) square wave. The phase response is zero for all components. The fundamental period is $T = 2$ and therefore the fundamental frequency is $f_0 = 1/2 = 0.5\text{ Hz}$ and harmonics are therefore 0.5 Hz apart when the Fourier series is calculated.

For the above square wave we can calculate the Fourier series using Eq. 27 as:

$$C_0 = \frac{1}{T}\int_0^T s(t)dt = \frac{1}{2}\int_0^1 1\,dt = \frac{t}{2}\bigg|_0^1 = \frac{1}{2} \tag{28}$$

$$C_n = \frac{1}{T}\int_0^T s(t)e^{-j\omega_0 nt}dt = \frac{1}{2}\int_0^1 e^{-j\pi nt}dt = \left.\frac{e^{-j\pi nt}}{-2j\pi n}\right|_0^1 = \frac{e^{-j\pi n}-1}{-2j\pi n}$$

$$= \left(\frac{e^{\frac{j\pi n}{2}} - e^{\frac{-j\pi n}{2}}}{2j\pi n}\right)e^{\frac{-j\pi n}{2}} = \frac{\sin \pi n/2}{\pi n}e^{\frac{-j\pi n}{2}} \tag{29}$$

recalling that $\sin x = (e^{jx} - e^{-jx})/2j$

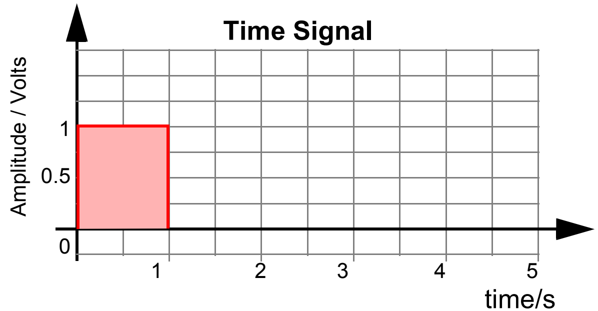
Noting that $e^{-j\pi n/2} = \cos \pi n/2 - j\sin \pi n/2 = 0, j$ or $-j$ (depending on the value of n) and recalling from Eq. 16 and 17 (see *Fourier Series*) that $C_n = A_n + jB_n$ then the square wave can be decomposed into a sum of harmonically related sine waves of amplitudes:

$$A_0 = 1/2$$

$$A_n = \begin{cases} 1/n\pi \text{ for odd } n \\ 0 \text{ for even } n \end{cases} \tag{30}$$

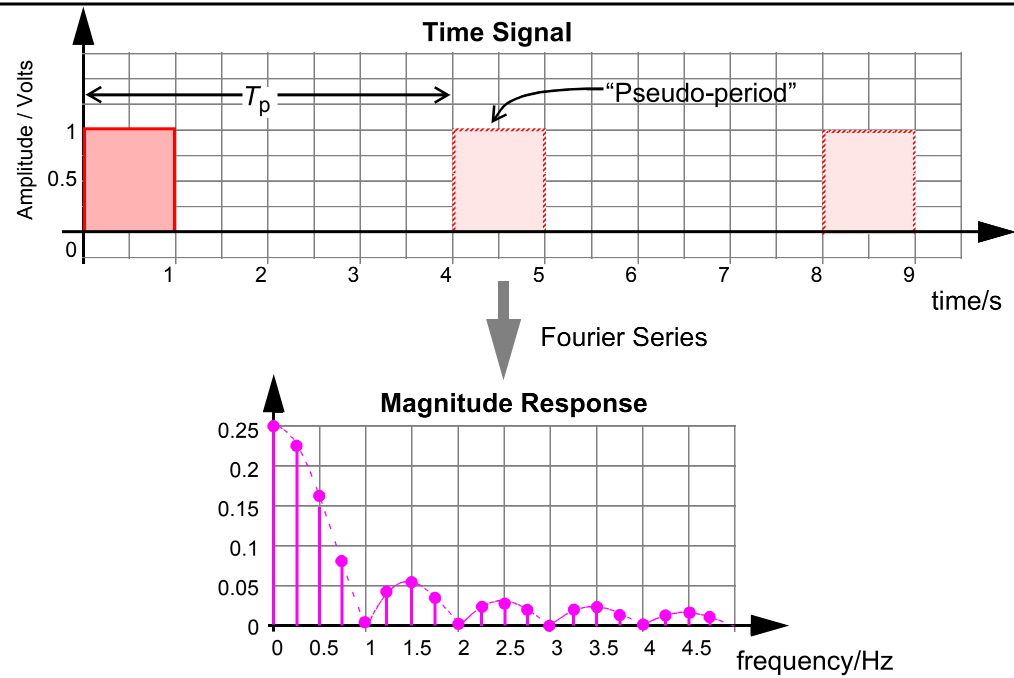
The amplitude response of the Fourier series is plotted above.

Now consider the case where the signal is aperiodic, and is in fact just a single pulse:



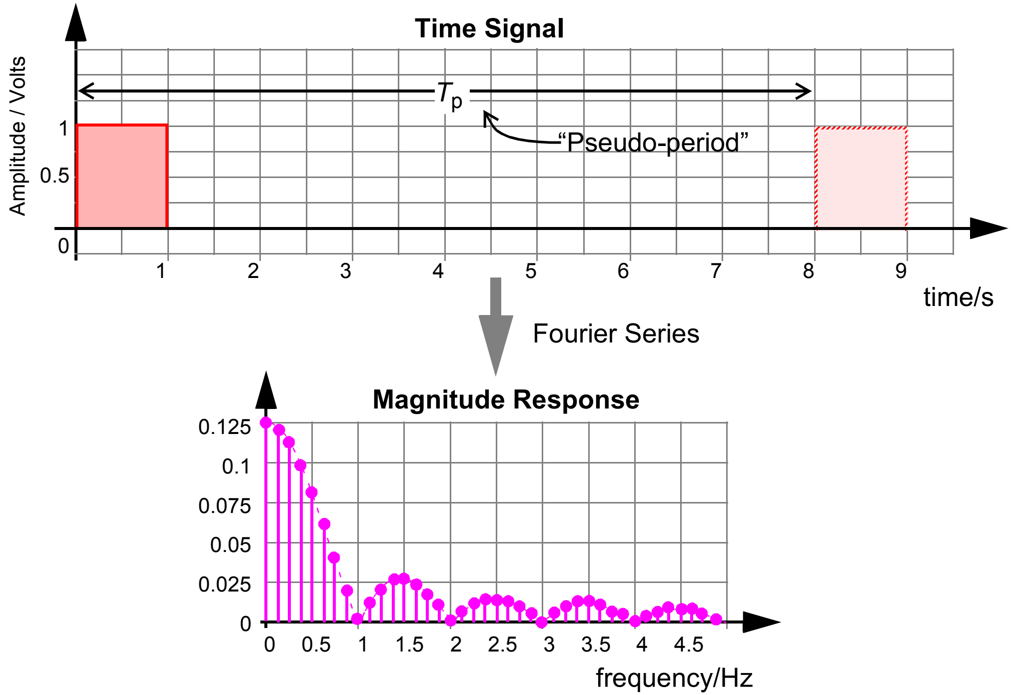
A single aperiodic pulse. This signal is most definitely not periodic and therefore the Fourier series cannot be calculated.

One way to obtain "some" information on the sinusoidal components comprising this aperiodic signal would be to assume the existence of a periodic "relative" or "pseudo-period" of this signal:



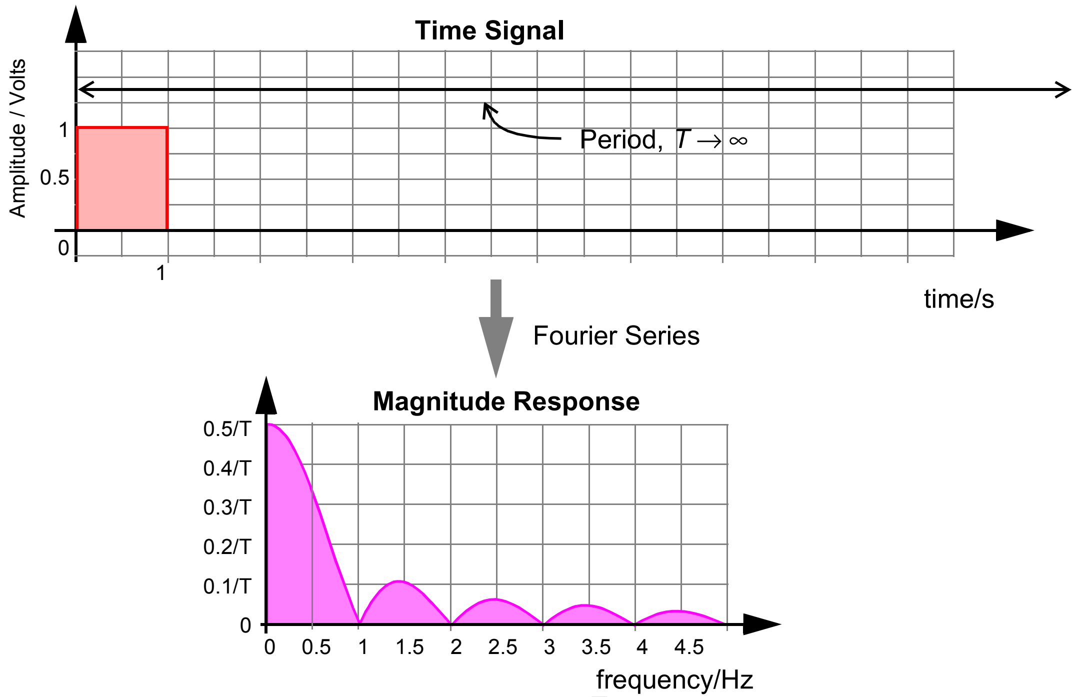
A periodic signal that is clearly a relative of the single pulse aperiodic signal. By adding the pseudo-periods we essentially assume that the single pulse of interest is a periodic signal and therefore we can now use the Fourier series tools to analyse. The fundamental period, $T_p = 4$ and therefore the harmonics of the Fourier series are placed $f_0 = 0.25$ Hz apart.

If we assumed that the “periodicity” of the pulse was even longer, say 8 seconds, then the spacing between the signal harmonics would further decrease:



If we increase the fundamental pseudo-period to $T_p = 8$ the harmonics of the Fourier series are more closely spaced at $f_0 = 1/8 = 0.125$ Hz apart. The magnitude of all the harmonics proportionally decreases with the increase in the pseudo-period. This is expected since the power of the signal decreases as the number of harmonics decreases.

If we further assumed that the period of the signal was such that $T \to \infty$ then $f_0 \to 0$ and given the finite energy in the signal, the magnitude of each of the Fourier series sine waves will tend to zero given that the harmonics are now so closely spaced! Hence if we multiply the magnitude response by T and plot the Fourier series we have now realised a graphical interpretation of the Fourier transform:



If we increase the fundamental pseudo-period such that $T \to \infty$ the frequency spacing between the harmonics of the Fourier series tends to zero, i.e. $f_0 \to 0$. Note that the magnitude of the Fourier series components are scaled proportionally down by the value of the "pseudo" period and in the limit as $T \to \infty$ will tend to zero. Hence the y-axis is plotted as $1/T$.

To realizethe mathematical version of the Fourier transform first define a new function based on the general Fourier series of Eq. 27 such that:

$$X(f) = \frac{C_n}{f_0} = C_n T \tag{31}$$

then:

$$x(t) = \sum_{n=-\infty}^{\infty} C_n e^{j2\pi n f_0 t}$$
$$X(f) = \int_{-T/2}^{T/2} x(t) e^{-j2\pi n f_0 t}\, dt = \int_{-\infty}^{\infty} x(t) e^{-j2\pi f t} dt \tag{32}$$

where nf_0 becomes the continuous variable f as $f_0 \to 0$ and $n \to \infty$. This equation is referred to as the Fourier transform and can of course be written in terms of the angular frequency:

$$X(\omega) = \int_{-\infty}^{\infty} x(t) e^{-j\omega t} dt \tag{33}$$

Knowing the Fourier transform of a signal, of course allows us to transform back to the original aperiodic signal:

$$x(t) = \sum_{n=-\infty}^{\infty} C_n e^{j2\pi n f_0 t} = \sum_{n=-\infty}^{\infty} X(f) f_0 e^{j2\pi n f_0 t} = \left(\sum_{n=-\infty}^{\infty} X(f) e^{j2\pi n f_0 t} \right) f_0$$
$$\Rightarrow x(t) = \int_{-\infty}^{\infty} X(f) e^{j2\pi f t}\, df \tag{34}$$

This equation is referred to as the inverse Fourier transform and can also be written in terms of the angular frequency:

$$x(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega) e^{j\omega t}\, d\omega \tag{35}$$

Hence we have realised the Fourier transform analysis and synthesis pair of equations:

$$x(t) = \int_{-\infty}^{\infty} X(f) e^{j2\pi f t}\, df \quad \text{Synthesis}$$
$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi f t} dt \quad \text{Analysis} \tag{36}$$

Fourier Transform Pair

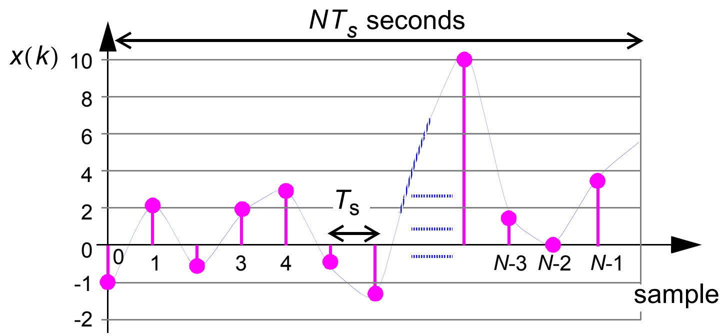
Therefore the Fourier transform of a continuous time signal, $x(t)$, will be a continuous function in frequency.

2.5 Discrete Fourier Transform

From the Fourier transform pair above, in the case where a signal is sampled at intervals of T_s seconds and is therefore discrete, the Fourier transform analysis equation will become:

$$X(f) = \int_{-\infty}^{\infty} x(kT_s)e^{-j2\pi fkT_s}d(kT_s) \tag{37}$$

where k is the time index.



Sampling an analog signal, $x(t)$, to produce a discrete time signal, $x(kT_s)$ written as $x(k)$. The sampling period is T_s and the sampling frequency is therefore $f_s = 1/T_s$. The total time duration of the K samples is KT_s seconds. Just as there exists a continuous time Fourier transform, we can also derive a discrete Fourier transform (DFT) in order to assess what sinusoidal frequency components comprise this signal.

and hence we can write:

$$X(f) = \sum_{k=-\infty}^{\infty} x(kT_s)e^{-j2\pi fkT_s} = \sum_{k=-\infty}^{\infty} x(kT_s)e^{\frac{-j2\pi fk}{f_s}} \tag{38}$$

where $f_s = 1/T_s$ is the sampling frequency. To further simplify we can write the discrete time signal simply in terms of its sample number:

$$X(f) = \sum_{k=-\infty}^{\infty} x(kT_s)e^{-j2\pi fkT_s} = \sum_{k=-\infty}^{\infty} x(k)e^{\frac{-j2\pi fk}{f_s}} \tag{39}$$

where $x(k)$ represents the kth sample of $x(t)$, which corresponds to $x(t = kT_s)$. Of course, if our signal is causal then the first sample is at $k = 0$, and the last sample is at $k = K-1$, giving a total of K samples:

$$X(f) = \sum_{k=0}^{K-1} x(k)e^{\frac{-j2\pi fk}{f_s}} \tag{40}$$

By using a finite number of data points this also forces the implicit assumption that our signal is now periodic, with a period of K samples, or KT_s seconds (see above figure). Therefore, noting that Eq. 40 is actually calculated for a continuous frequency variable, f, then in actual fact we need only evaluate this equation at specific frequencies which are the zero frequency (DC) and harmonics of the "fundamental" frequency, $f_0 = 1/(KT_s) = f_s/K$, i.e. K discrete frequencies of 0, f_0, $2f_0$, up to $(K-1)f_0$.

$$X\left(\frac{nf_s}{K}\right) = \sum_{k=0}^{K-1} x(k)e^{\frac{-j2\pi nf_sk}{Kf_s}} \quad \text{for } n = 0 \text{ to } K-1 \tag{41}$$

Simplifying to use only the time index, k, and the frequency index, n, gives the *discrete Fourier transform:*

$$X(n) = \sum_{k=0}^{K-1} x(k) e^{\frac{-j2\pi nk}{K}} \quad \text{for } n = 0 \text{ to } K-1 \tag{42}$$

If we recall that the discrete signal $x(k)$ was sampled at f_s then the signal has image (or alias) components above $f_s/2$, then when evaluating Eq. 42 it is only necessary to evaluate up to $f_s/2$, and therefore the DFT is further simplified to:

$$\boxed{X(n) = \sum_{k=0}^{K-1} x(k) e^{\frac{-j2\pi nk}{K}} \quad \text{for } n = 0 \text{ to } K/2} \tag{43}$$

Discrete Fourier Transform

Clearly because we have evaluated the DFT at only K frequencies, then the frequency resolution is limited to the DFT “bins” of frequency width f_s/K Hz.

Note that the discrete Fourier transform only requires multiplications since each complex exponential is computed in its complex number form.

$$e^{\frac{-j2\pi nk}{K}} = \cos\frac{2\pi nk}{K} - j\sin\frac{2\pi nk}{K} \tag{44}$$

If the signal $x(k)$ is real valued, then the DFT computation requires approximately K^2 real multiplications and adds (noting that a real value multiplied by a complex value requires two real multiplies). If the signal $x(k)$ is complex then a total of $2K^2$ MACs are required (noting that the multiplication of two complex values requires four real multiplications).

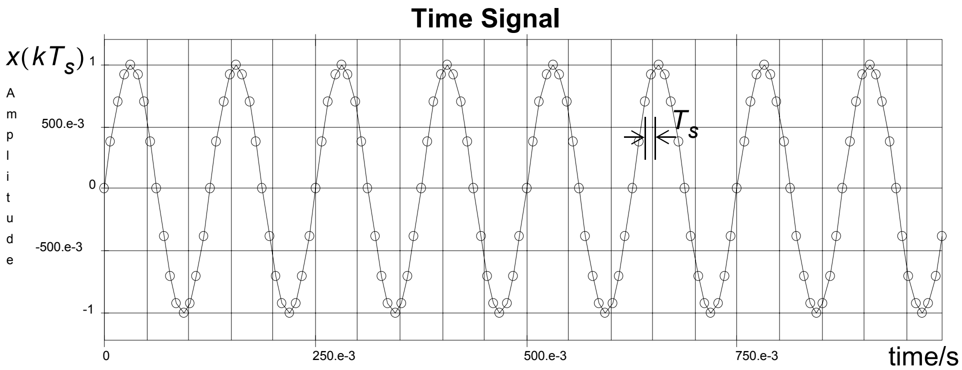
From the DFT we can calculate a magnitude and a phase response:

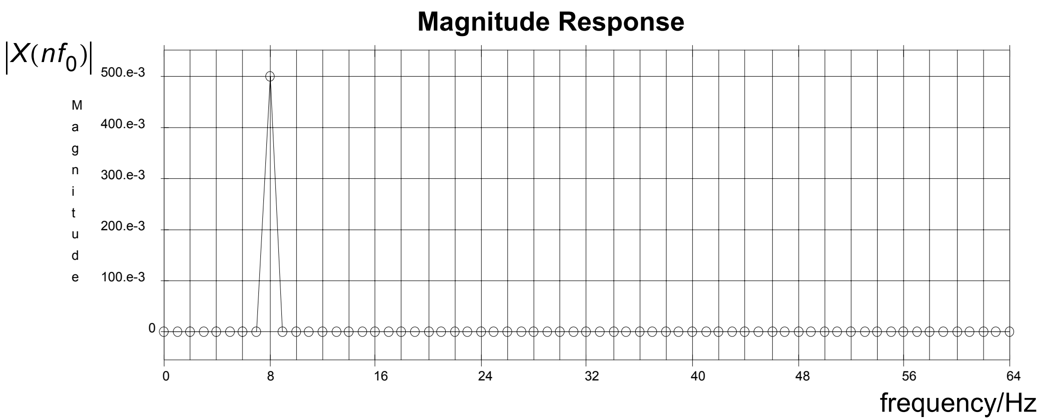
$$X(n) = |X(n)| \angle X(n) \tag{45}$$

From a given DFT sequence, we can of course calculate the inverse DFT from:

$$x(k) = \frac{1}{K} \sum_{n=0}^{K-1} X(n) e^{\frac{j2\pi nk}{K}} \tag{46}$$

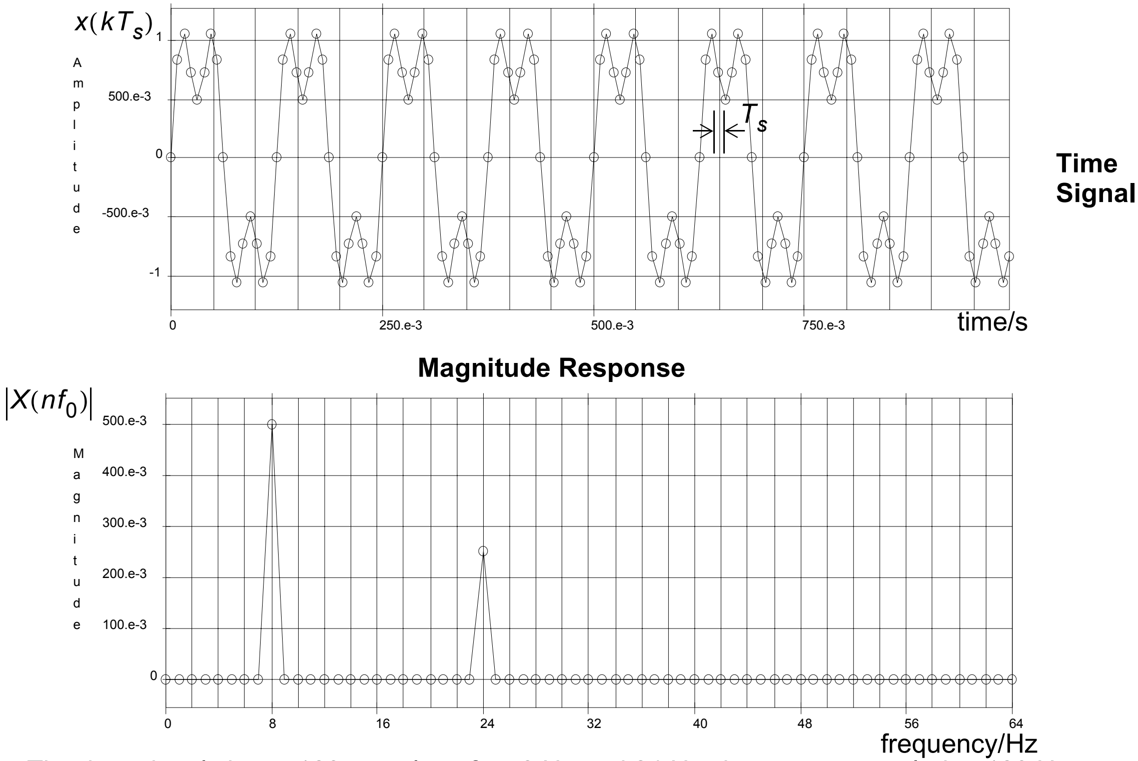
As an example consider taking the DFT of 128 samples of an 8 Hz sine wave sampled at 128 Hz:





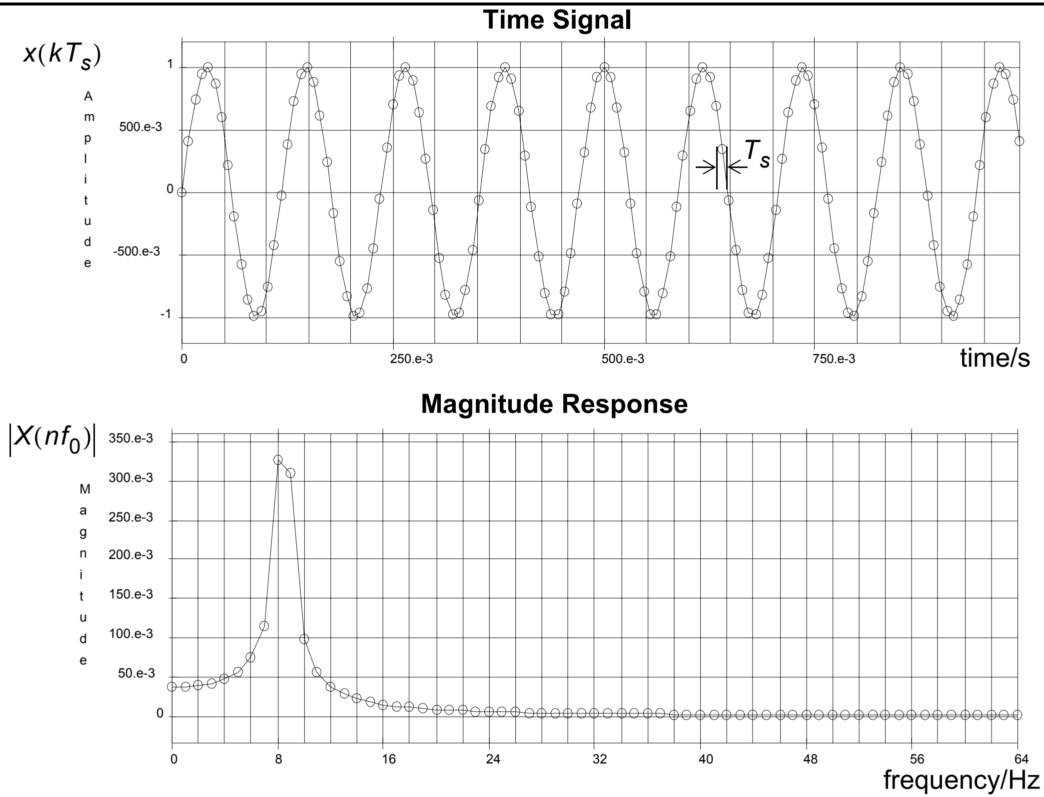
The time signal shows 128 samples of an 8 Hz sine wave sampled at 128 Hz: $x(k) = \sin(16\pi k)/128$. Note that there are exactly an integral number of periods (eight) present over the 128 samples. Taking the DFT exactly identifies the signal as an 8 Hz sinusoid. The DFT magnitude spectrum has an equivalent negative frequency portion which is identical to that of the positive frequencies if the time signal was real valued.

If we take the DFT of the slightly more complex signal consisting of an 8 Hz and a 24 Hz sine wave of half the amplitude of the 8 Hz then:



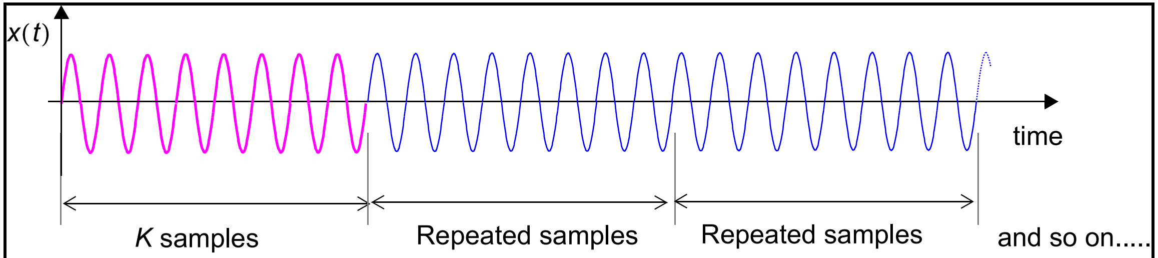
The time signal shows 128 samples of an 8 Hz and 24 Hz sine waves sampled at 128 Hz: $x(k) = \sin(16\pi k)/128 + 0.5\sin(48\pi k)/128$. Note that there are exactly an integral number of periods present for both sinusoids over the 128 samples.

Now consider taking the DFT of 128 samples of an 8.5 Hz sine wave sampled at 128 Hz:



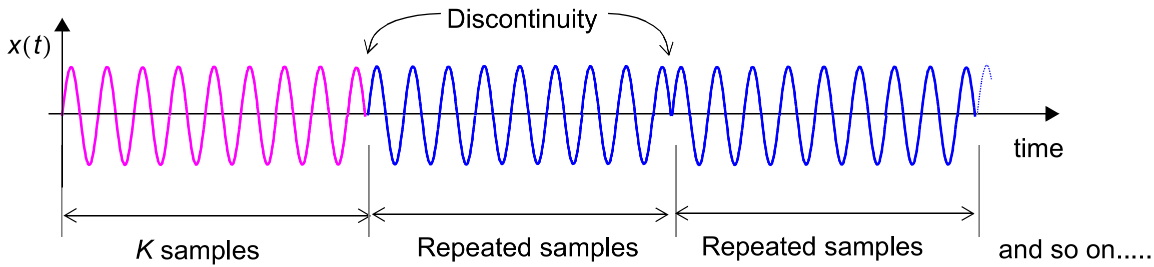
The time signal shows 128 samples of an 8.5 Hz sine wave sampled at 128 Hz: $x(k) = \sin(17\pi k)/128$. Note that because the 8.5Hz sine wave does not lie exactly on a frequency bin, then its energy appears spread over a number of frequency bins around 8 Hz.

So why is the signal energy now spread over a number of frequency bins? We can interpret this by recalling that the DFT implicitly assumes that the signal is periodic, and the K data points being analysed are one full period of the signal. Hence the DFT assumes the signal has the form:



If there are an integral number of sine wave periods in the K samples input to the DFT computation, then the spectral peaks will fall exactly on one of the frequency bins as shown earlier. Essentially the result produced for the DFT computation has assumed that the signal was periodic, and the K samples form one period of the signal and thereafter the period repeats. Hence the DFT assumes the *complete* signal is as illustrated above (the discrete samples are not shows for clarity.

If there are not an integral number of periods in the signal (as for the 8.5 Hz example), then:



If there are not an integral number of sine wave periods in the K samples input to the DFT computation, then the spectral peaks will not fall exactly on one of the frequency bins. As the DFT computation has assumed that the signal was periodic, the DFT interprets that the signal undergoes a “discontinuity” jump at the end of the K samples. Hence the result of the DFT interprets the time signal as if this discontinuity was part of it. Hence more than one single sine wave is required to produce this waveform and thus a number of frequency bins indicate sine wave components being present.

2.6 DFT Spectral Aliasing

Note that the discrete Fourier transform of a signal $x(k)$ is periodic in the frequency domain. If we assume that the signal was real and was sampled above the Nyquist rate f_s, then there are no frequency components of interest above $f_s/2$. From the Fourier transform, if we calculate the frequency components up to frequency $f_s/2$ then this is equivalent to evaluating the DFT for the first $K/2-1$ discrete frequency samples:

$$X(n) = \sum_{k=0}^{K-1} x(k)e^{\frac{-j2\pi nk}{K}} \quad \text{for } n = 0 \text{ to } K/2 - 1 \tag{47}$$

Of course if we evaluate for the next $K/2 - 1$ discrete frequencies (i.e. from $f_s/2$ to f_s) then:

$$X(n) = \sum_{k=0}^{K-1} x(k)e^{\frac{-j2\pi nk}{K}} \quad \text{for } n = K/2 \text{ to } K - 1 \tag{48}$$

In Eq. 47 if we substitute for the variable $i = K - n \Rightarrow n = K - i$ and calculate over range $i = 1$ to $K/2$ (equivalent to the range $n = K/2$ to $K - 1$) then:

$$X(i) = \sum_{k=0}^{K-1} x(k)e^{\frac{-j2\pi ik}{K}} \quad \text{for } i = 1 \text{ to } K/2 \tag{49}$$

and we can write:

$$
\begin{aligned}
X(K-n) &= \sum_{k=0}^{K-1} x(k) e^{\frac{-j2\pi(K-n)k}{K}} \\
&= \sum_{k=0}^{K-1} x(k) e^{\frac{j2\pi nk}{K}} e^{\frac{-j2\pi Kk}{K}} = \sum_{k=0}^{K-1} x(k) e^{\frac{j2\pi nk}{K}} e^{-j2\pi k} \\
&= \sum_{k=0}^{K-1} x(k) e^{\frac{j2\pi kn}{K}} \quad \text{for } n = K/2 \text{ to } K-1
\end{aligned}
\tag{50}
$$

since $e^{-j2\pi k} = 1$ for all integer values of k. Therefore from Eq. 50 it is clear that:

$$
|X(n)| = |X(K-n)| \tag{51}
$$

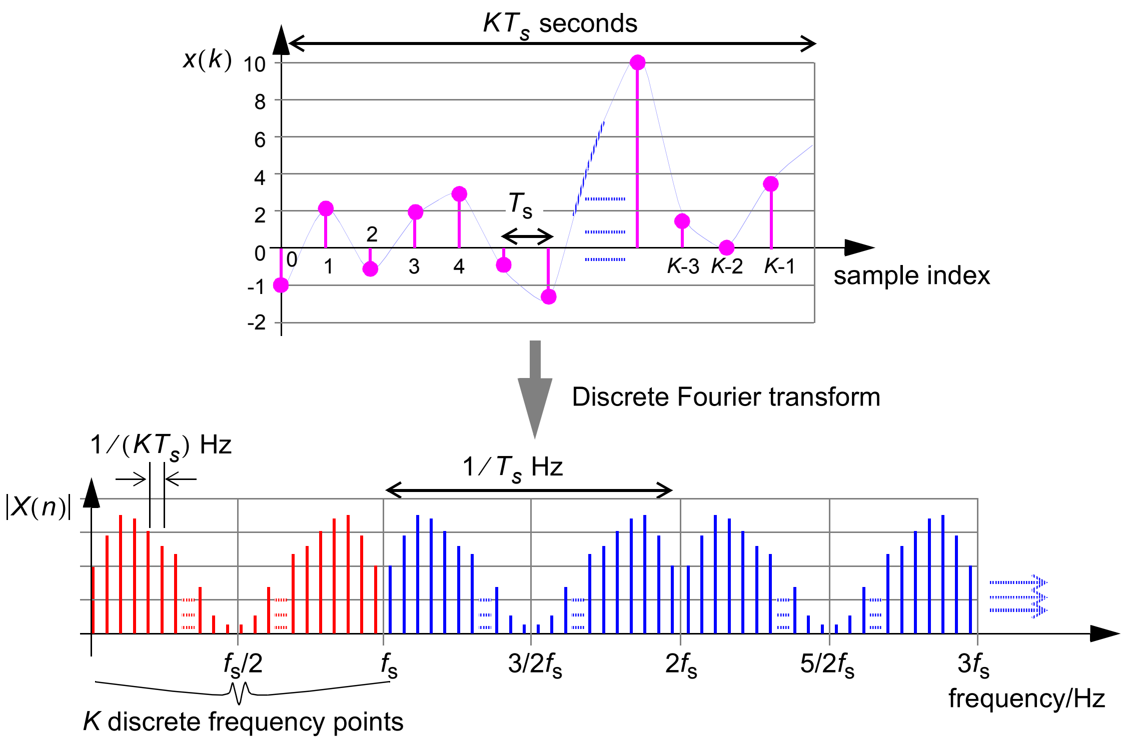
Hence when we plot the DFT it is symmetrical about the $K/2$ frequency sample, i.e. the frequency value $f_s/2$ Hz depending on whether we plot the x-axis as a frequency index or a true frequency value.

We can further easily show that if we take a value of frequency index n above $K-1$ (i.e. evaluate the DFT above frequency f_s, then:

$$
\begin{aligned}
X(n+mK) &= \sum_{k=0}^{K-1} x(k)e^{\frac{-j2\pi(n+mK)k}{K}} = \sum_{k=0}^{K-1} x(k)e^{\frac{-j2\pi nk}{K}}e^{-j2\pi mk} \\
&= \sum_{k=0}^{K-1} x(k)e^{\frac{-j2\pi nk}{K}} \\
&= X(n)
\end{aligned}
\tag{52}
$$

where m is a positive integer and we note that $e^{-j2\pi mk} = 1$.

Therefore we can conclude that when evaluating the magnitude response of the DFT the components of specific interest cover the (baseband) frequencies from 0 to $f_s/2$, and the magnitude spectra will be symmetrical about the $f_s/2$ line and periodic with period f_s:



Spectral aliasing. The main portion of interest of the magnitude response is the “baseband” from 0 to $f_s/2$ Hz. The “baseband” spectra is symmetrical about the point $f_s/2$ and thereafter periodic with period f_s Hz.

2.7 Fast Fourier Transform (FFT)

The FFT is a method of computing the discrete Fourier transform (DFT) that exploits the redundancy in the general DFT equation:

$$X(n) = \sum_{k=0}^{K-1} x(k) e^{\frac{-j2\pi nk}{K}} \quad \text{for } n = 0 \text{ to } K-1 \tag{53}$$

Noting that the DFT computation of Eq. 53 requires approximately K^2 complex multiply accumulates (MACs), where K is a power of 2, the radix-2 FFT requires only $K\log_2 K$ MACs. The computational savings achieved by the FFT is therefore a factor of $K/\log_2 K$. When K is large this saving can be considerable. The following table compares the number of MACs required for different values of K for the DFT and the FFT:

K	***DFT MACs***	***FFT MACs***
32	1024	160
1024	1048576	10240
32768	~ 1 x 10^9	~ 0.5 x10^6

There are a number of different FFT algorithms sometimes grouped via the names Cooley-Tukey, prime factor, decimation-in-time, decimation-in-frequency, radix-2 and so on. The bottom line for all FFT algorithms is, however, that they remove redundancy from the direct DFT computational algorithm of Eq. 53.

We can highlight the existence of the redundant computation in the DFT by inspecting Eq. 53. First, for notational simplicity we can rewrite Eq. 53 as:

$$X(n) = \sum_{k=0}^{K-1} x(k) W_K^{-nk} \quad \text{for } n = 0 \text{ to } K-1 \tag{54}$$

where $W_K = e^{j2\pi/K} = \cos 2\pi/K + j\sin 2\pi/K$. Using the DFT algorithm to calculate the first four components of the DFT of a (trivial) signal with only 8 samples requires the following computations:

$$\begin{aligned}
X(0) &= x(0) + x(1) + x(2) + x(3) + x(4) + x(5) + x(6) + x(7) \\
X(1) &= x(0) + x(1)W_8^{-1} + x(2)W_8^{-2} + x(3)W_8^{-3} + x(4)W_8^{-4} + x(5)W_8^{-5} + x(6)W_8^{-6} + x(7)W_8^{-7} \\
X(2) &= x(0) + x(1)W_8^{-2} + x(2)W_8^{-4} + x(3)W_8^{-6} + x(4)W_8^{-8} + x(5)W_8^{-10} + x(6)W_8^{-12} + x(7)W_8^{-14} \\
X(3) &= x(0) + x(1)W_8^{-3} + x(2)W_8^{-6} + x(3)W_8^{-9} + x(4)W_8^{-12} + x(5)W_8^{-15} + x(6)W_8^{-18} + x(7)W_8^{-21}
\end{aligned} \tag{55}$$

However note that there is redundant (or repeated) arithmetic computation in Eq. 55. For example, consider the third term in the second line of Eq. 55:

$$x(2)W_8^{-2} = x(2)e^{j2\pi\left(\frac{-2}{8}\right)} = x(2)e^{\frac{-j\pi}{2}} \tag{56}$$

Now consider the computation of the third term in the fourth line of Eq. 55:

$$x(2)W_8^{-6} = x(2)e^{j2\pi\left(\frac{-6}{8}\right)} = x(2)e^{\frac{-j3\pi}{2}} = x(2)e^{-j\pi}e^{\frac{-j\pi}{2}} = -x(2)e^{\frac{-j\pi}{2}} \tag{57}$$

Therefore we can save one multiply operation by noting that the term $x(2)W_8^{-6} = -x(2)W_8^{-2}$. In fact because of the periodicity of W_K^{nk} every term in the fourth line of Eq. 55 is available from the computed terms in the second line of the equation. Hence a considerable saving in multiplicative computations can be achieved if the computational order of the DFT algorithm is carefully considered.

More generally we can show that the terms in the second line of Eq. 55 are:

$$x(k)W_8^{-k} = x(k)e^{\frac{-j2\pi k}{8}} = x(k)e^{\frac{-j\pi k}{4}} \tag{58}$$

and for terms in the fourth line of Eq. 55:

$$\begin{aligned} x(k)W_8^{-3k} &= x(k)e^{\frac{-j6\pi k}{8}} = x(k)e^{\frac{-j3\pi k}{4}} = x(k)e^{-j\left(\frac{\pi}{2}+\frac{\pi}{4}\right)k} = x(k)e^{-j\frac{\pi k}{2}}e^{-j\frac{\pi k}{4}} \\ &= x(k)(-j)^k e^{-j\frac{\pi k}{4}} \\ &= (-j)^k x(k)W_8^{-k} \end{aligned} \tag{59}$$

This exploitation of the computational redundancy is the basis of the FFT which allows the same result as the DFT to be computed, but with less MACs.

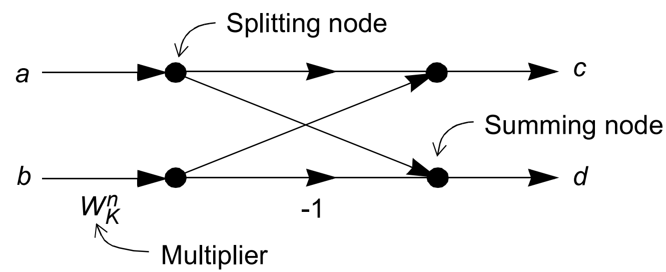
To more formally derive one version of the FFT (decimation-in-time radix-2), consider splitting the DFT equation into two “half signals” consisting of the odd numbered and even numbered samples, where the total number of samples is a power of 2 ($K = 2^k$):

$$
\begin{aligned}
X(n) &= \sum_{k=0}^{K/2-1} x(2k)e^{\frac{-j2\pi n(2k)}{K}} + \sum_{k=0}^{K/2-1} x(2k+1)e^{\frac{-j2\pi n(2k+1)}{K}} \\
&= \sum_{k=0}^{K/2-1} x(2k)W_K^{-2kn} + \sum_{k=0}^{K/2-1} x(2k+1)W_K^{-(2k+1)n} \\
&= \sum_{k=0}^{K/2-1} x(2k)W_K^{-2kn} + W_K^{-n} \sum_{k=0}^{K/2-1} x(2k+1)W_K^{-2kn}
\end{aligned}
\tag{60}
$$

Notice in Eq. 60 that the K point DFT which requires K^2 MACs in Eq. 53 is now accomplished by performing two $K/2$ point DFTs requiring a total of $2 \times K^2/4$ MACs which is a computational saving of 50%. Therefore a next logical step is to take the $K/2$ point DFTs and perform as $K/4$ point DFTs, saving 50% computation again, and so on. As the number of points we started with was a power of 2, then we can perform this decimation of the signal a total of K times, and each time reduce the total computation of each stage to that of a “butterfly” operation. If $K = 2^k$ then the computational saving is a factor of:

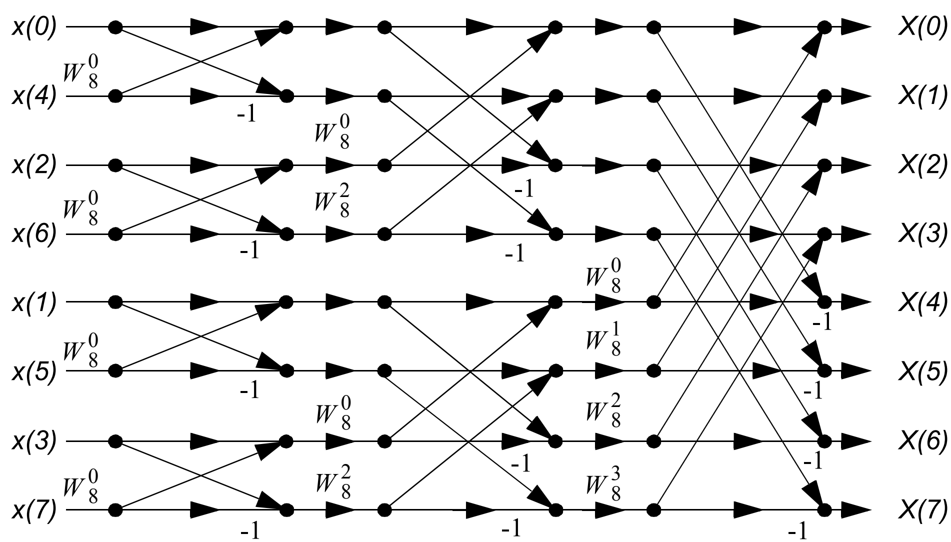
$$
\frac{K}{\log_2 K}
\tag{61}
$$

In general equations for an FFT are awkward to write mathematically, and therefore the algorithm is very often represented as a "butterfly" based signal flow graph (SFG), the butterfly being a simple signal flow graph of the form:



The butterfly signal flow graph. The multiplier W_K^n is a complex number, and the input data, a and b may also be complex. One butterfly computation requires one complex multiply and two complex additions (assuming the data is complex).

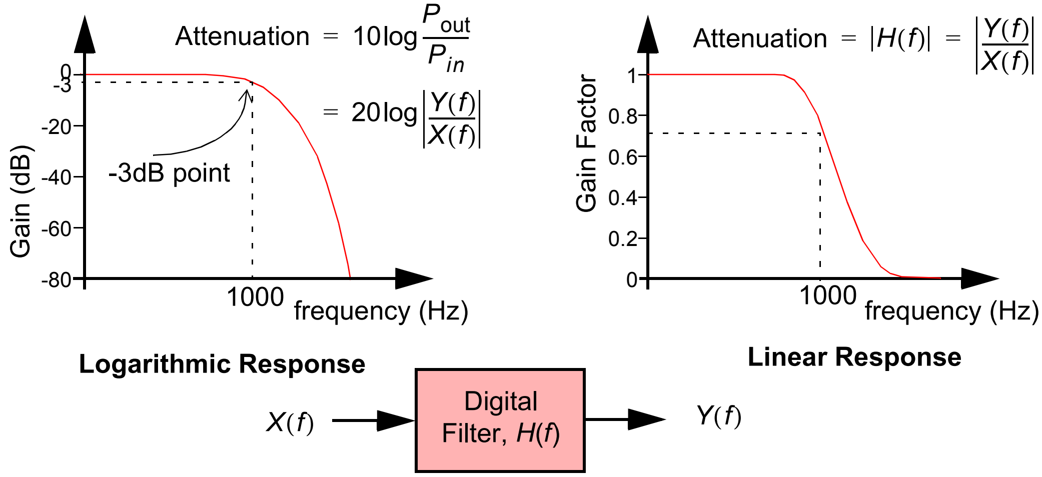
A more complete SFG for an 8 point decimation in time radix 2 FFT computation is:



A radix-2 Decimation-in-time (DIT) Cooley-Tukey FFT, for $K = 8$; $W_K^{nk} = e^{j2\pi nk/K}$.
Note that the butterfly computation is repeated through the SFG.

3 General Digital Filters

A DSP system that will filter a digital input (i.e., selectively discriminate signals in different frequency bands) according to some pre-designed criteria is called a *digital filter*. In some situations *digital filters* are used to modify phase. A *digital filter*'s characteristics are usually viewed via their frequency response and for some applications their phase response. For the frequency response, the filter attenuation or gain characteristic can either be specified on a linear gain scale, or more commonly a logarithmic gain scale:

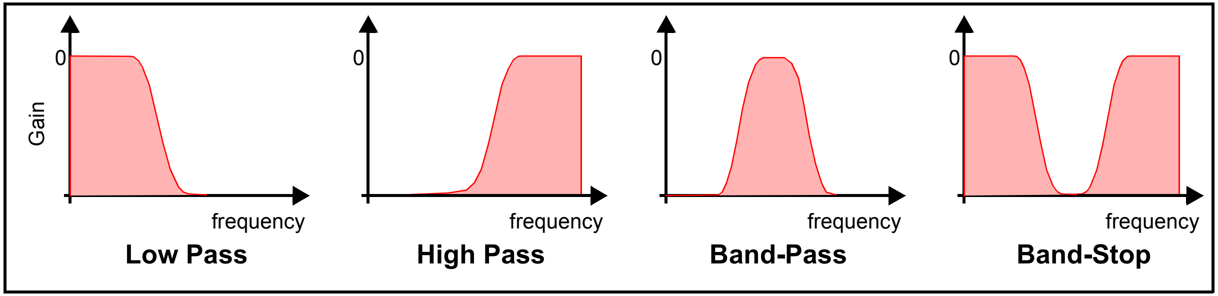


The above *digital filter* is a low pass filter cutting off at 1000 Hz. Both the linear and logarithmic magnitude responses of the transfer function, $H(f) = Y(f)/X(f)$ are shown. The cut-off frequency of a filter is usually denoted as the “3 dB frequency”, i.e. at f_{3dB} = 1000 Hz, the filter attenuates the power of a sinusoidal component signal at this frequency by 0.5, i.e.

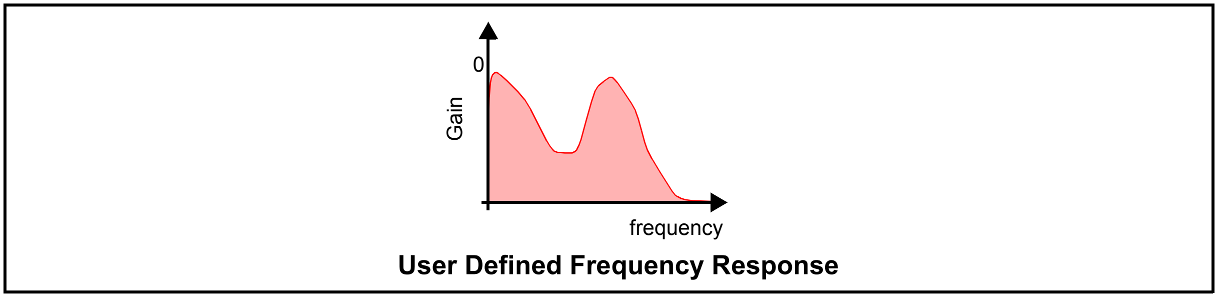
$$10\log\frac{P_{out}}{P_{in}}\bigg|_{f_{3\ dB}} = 20\log\left|\frac{Y(f_{3dB})}{X(f_{3dB})}\right| = 10\log 0.5 = 20\log 0.707\ldots = -3\text{ dB} \qquad (62)$$

The power of the output signal relative to the input signal at f_{3dB} is therefore 0.5, and the signal amplitude is attenuated by $1/\sqrt{2} = 0.707\ldots$. For a low pass filter signals with a frequency higher than f_{3dB} are attenuated by more than 3 dB.

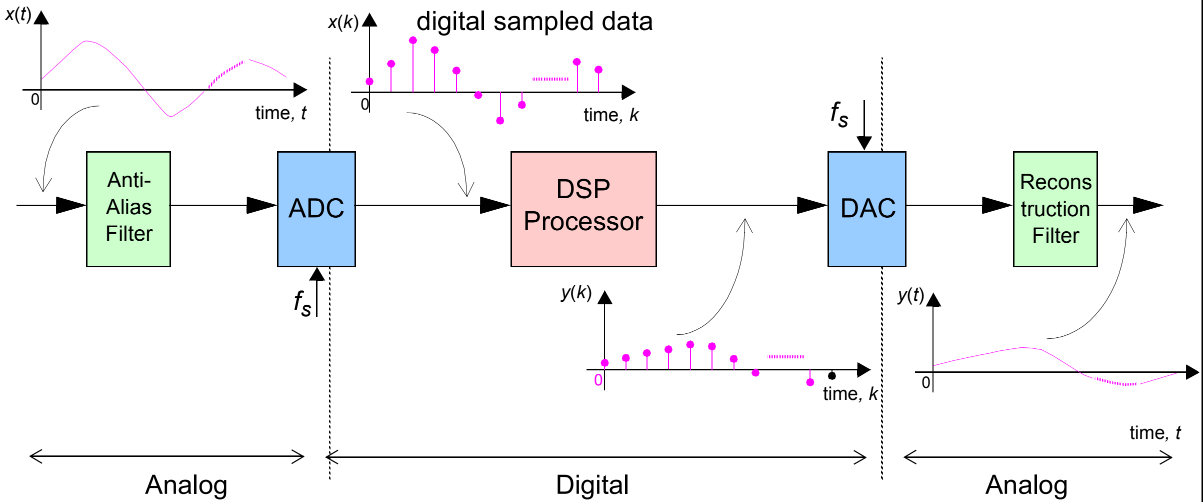
Digital filters are usually designed as either low pass, high pass, band-pass or band-stop:



SystemView will give the user the facility to design a filter for an arbitrary frequency response by "sketching" graphically the magnitude of its transfer function:



There are two types of linear digital filters, FIR (finite impulse response filter) and IIR (infinite impulse response filter). An *FIR* filter is a digital filter that performs a moving, weighted average on a discrete input signal, $x(n)$, to produce an output signal. The arithmetic computation required by the digital filter is of course performed on a DSP processor or equivalent:

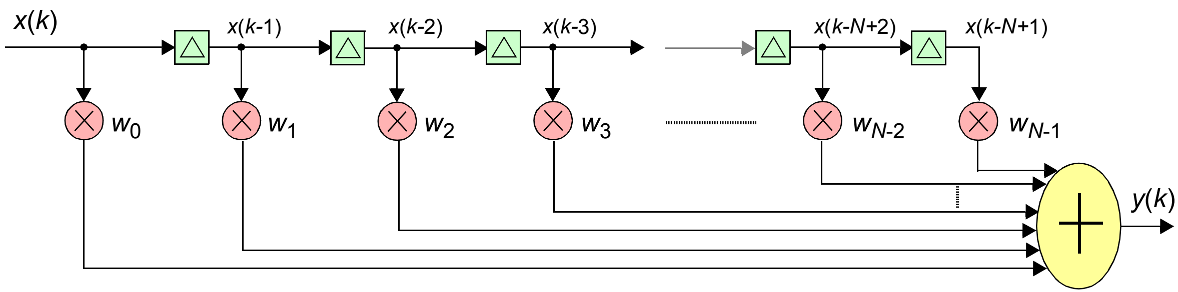


The digital filter equations are implemented on the DSP Processor which processes the time sampled data signal to produce a time sampled output data signal.

The actual frequency and phase response of the filter is found by taking the discrete frequency transform (DFT) of the weight values of w_0 to w_{N-1}.

4 Finite Impulse Response (FIR) Filter

An FIR digital filter can be conveniently represented in a signal flow graph:

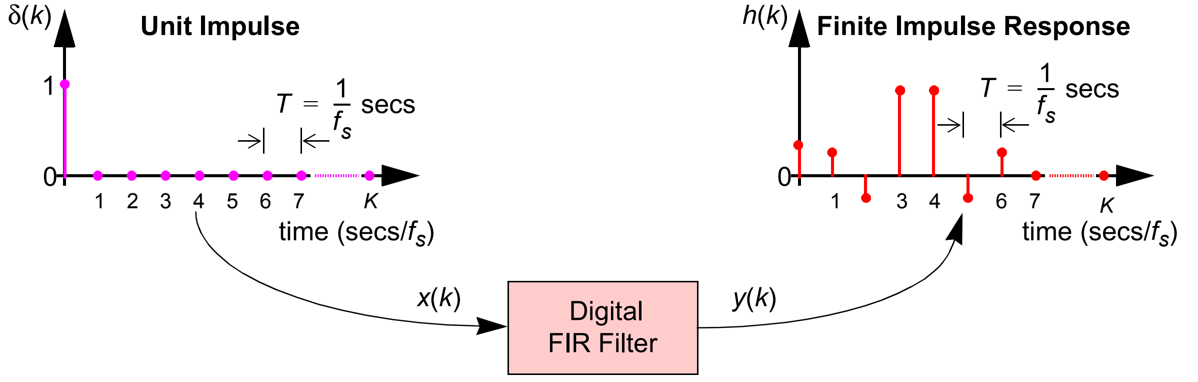


The signal flow graph and the output equation for an FIR *digital filter*. The last N input samples are weighted by the filter coefficients to produce the output $y(k)$

The general output equation (convolution) for an FIR filter is:

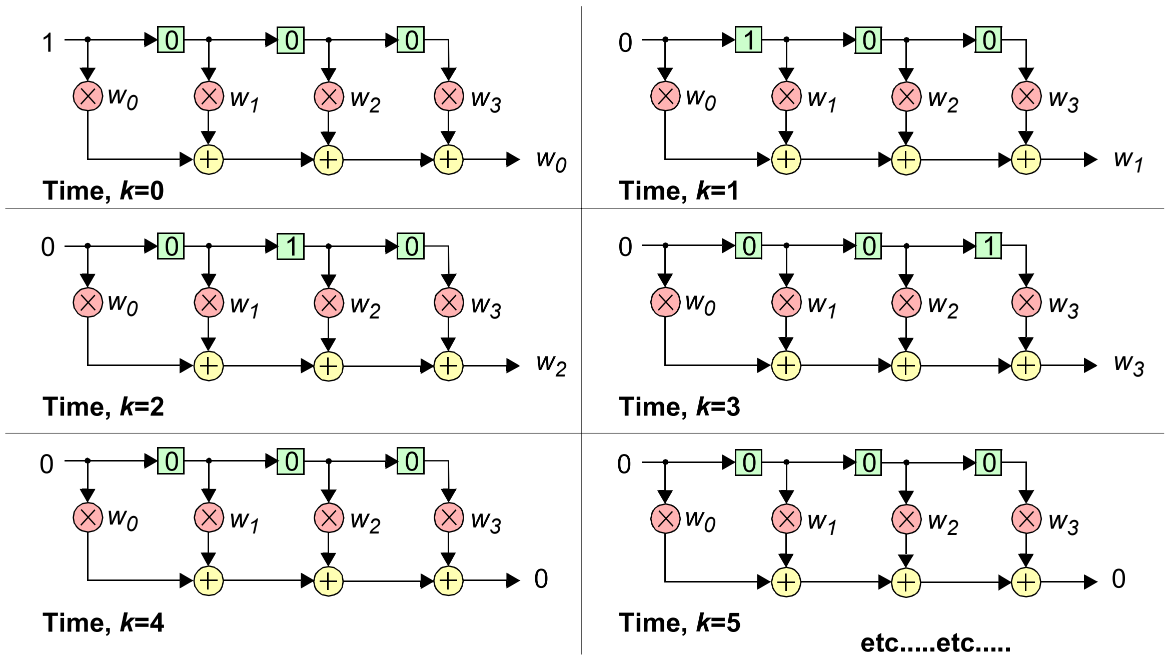
$$\begin{aligned} y(k) &= w_0 x(k) + w_1 x(k-1) + w_2 x(k-2) + w_3 x(k-3) + \ldots + w_{N-1} x(k-N+1) \\ &= \sum_{n=0}^{N-1} w_n x(k-n) \end{aligned} \tag{63}$$

The term finite impulse response refers to the fact that the impulse response results in energy at only a finite number of samples after which the output is zero. Therefore, if the input sequence is a unit impulse, the FIR filter output will have a finite duration:



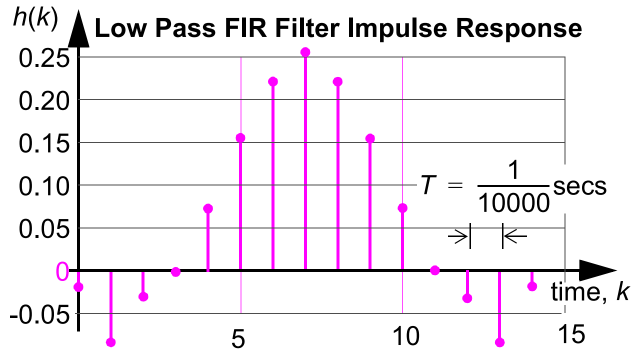
The discrete output of a finite impulse response (FIR) filter sampled at f_s Hz has a finite duration in time, i.e. the output will decay to zero within a finite time.

This can be illustrated by considering that the FIR filter is essentially a shift register which is clocked once per sampling period. For example consider a simple 4 weight filter:



When applying a unit impulse response to a filter, the 1 value passes through the filter "shift register" causing the filter impulse response to be output.

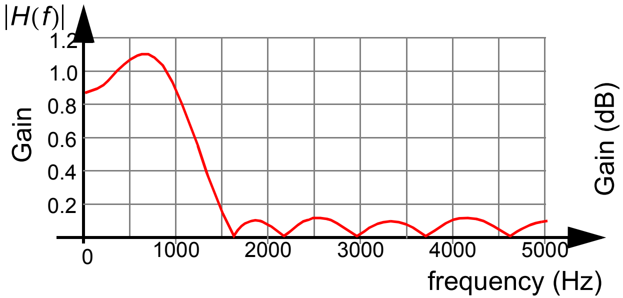
As an example, a simple low pass FIR filter can be designed using the DSP design software SystemView by Elanix, with a sampling rate of 10000 Hz, a cut off frequency of around 1000 Hz, a stopband attenuation of about 40 dB, passband ripple of less than 1 dB and limited to 15 weights. The resulting filter is:



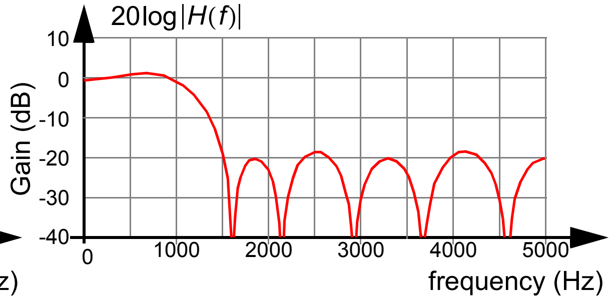
$w_0 = w_{14} = -0.01813...$
$w_1 = w_{13} = -0.08489...$
$w_2 = w_{12} = -0.03210...$
$w_3 = w_{11} = -0.00156...$
$w_4 = w_{10} = 0.07258...$
$w_5 = w_9 = 0.15493...$
$w_6 = w_8 = 0.22140...$
$w_7 = 0.25669...$
(Truncated to 5 decimal places)

The impulse response $h(k) = w_k$ of a low pass filter, **FIR1** with 15 weights, a sampling rate of 10000 Hz, and cut off frequency designed at around 1000Hz.

Noting that a unit impulse contains "all frequencies", then the magnitude frequency response and phase response of the filter are found from the DFT (or FFT) of the filter weights:

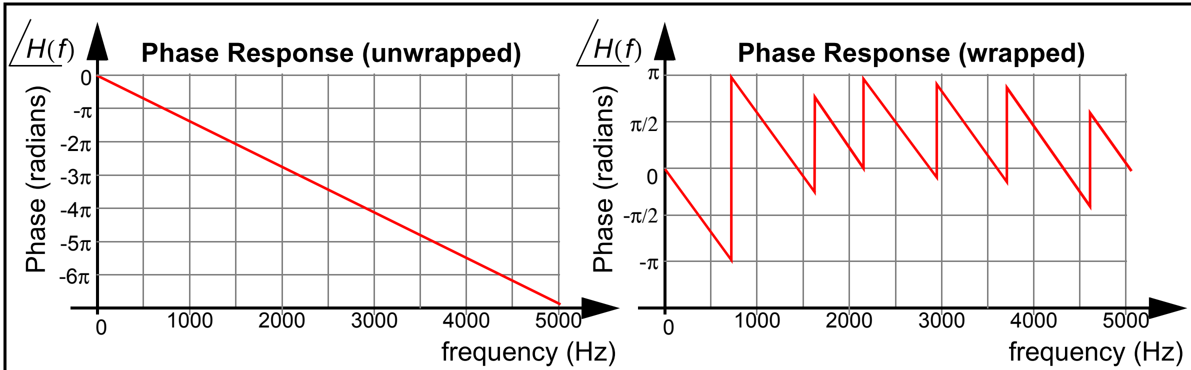


Linear Magnitude Response



Logarithmic Magnitude Response

The 1024 point FFT (zero padded) of the above low pass filter impulse response, **FIR1**. As the sampling rate is 10000 Hz the frequency response is only plotted up to 5000 Hz. (Note that the y-axis is labelled Gain rather than Attenuation, this is because -10 dB gain is the same as 10 dB attenuation. Hence if attenuation was plotted the above figures would be inverted.)

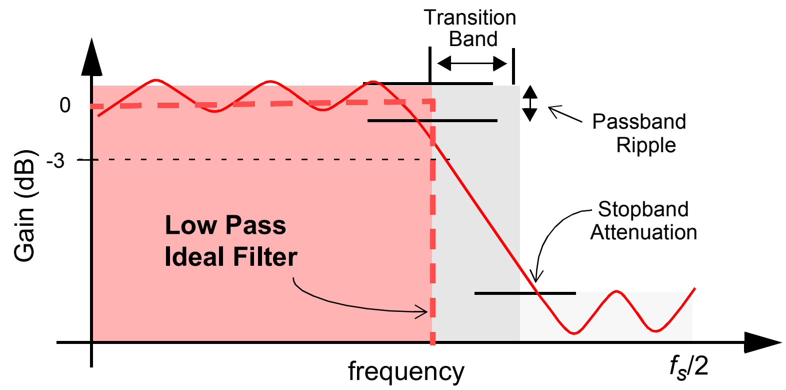


The 1024 point FFT generated phase response (phase shift versus frequency) above low pass filter impulse response, **FIR1**. Note that the filter is linear phase and the wrapped and unwrapped phase responses are different ways of representing the same information. The "wrapped" phase response will often produced by DSP software packages and gives phase values between $-\pi$ and π only. As the phase is calculated as modulo 2π. i.e. a phase shift of θ is the same as a phase shift of $\theta + 2\pi$ and so on. Phase responses are also often plotted using degrees rather than radians.

From the magnitude and phase response plots we can therefore calculate the attenuation and phase shift of different input signal frequencies. For example, if a single frequency at 1500 Hz, with an amplitude of 150 is input to the above filter, then the amplitude of the output signal will be around 30, and phase shifted by a little over -2π radians. However, if a single frequency of 500 Hz was input, then the output signal amplitude is amplified by a factor of about 1.085 and phase shifted by about -0.7π radians.

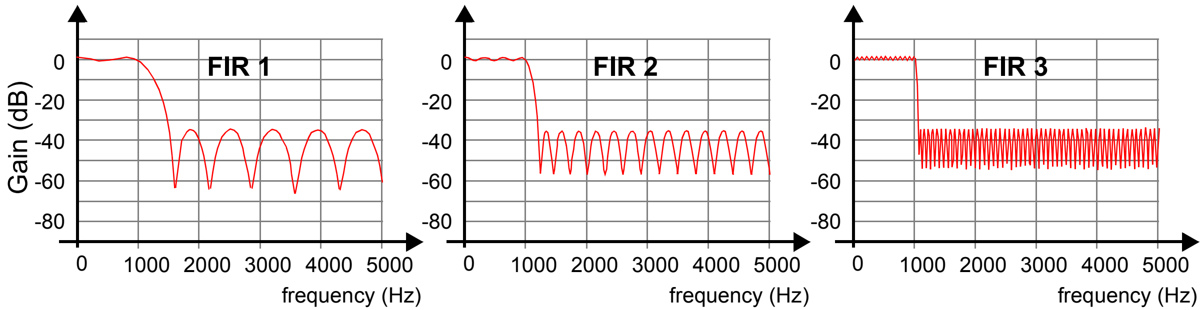
So, how long is a typical FIR filter? This of course depends on the requirement of the problem being addressed. For the generic filter characteristic shown below more weights are required if:

- A sharper transition bandwidth is required;
- More stopband attenuation is required;
- Very small passband ripple is required.



Generic low pass filter magnitude response. The more stringent the filter requirements of stopband attenuation, transition bandwidth and to a lesser extent passband ripple, the more weights that are required.

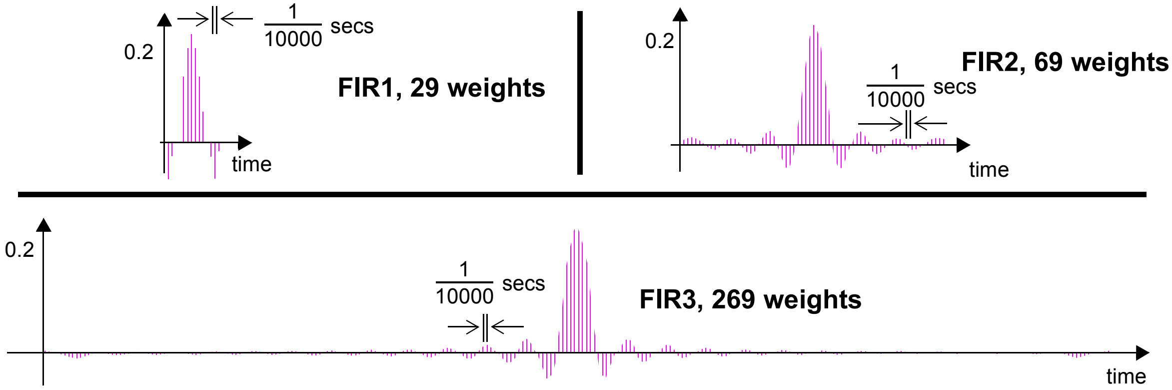
Consider again the design of the above FIR filter (FIR1) which was a low pass filter cutting of at about 1000 Hz. Using SystemView, the above criteria can be varied such that the number of filter weights can be increased and a more stringent filter designed. Consider the design of three low pass filters cutting off at 1000 Hz, with stopband attenuation of 40 dB and transition bandwidths 500 Hz, 200 Hz and 50 Hz:



Transition Band: 1000 - 1500Hz	**Transition Band: 1000 - 1200Hz**	**Transition Band: 1000 - 1100Hz**
No. of weights: 29	**No. of weights: 69**	**No. of weights: 269**

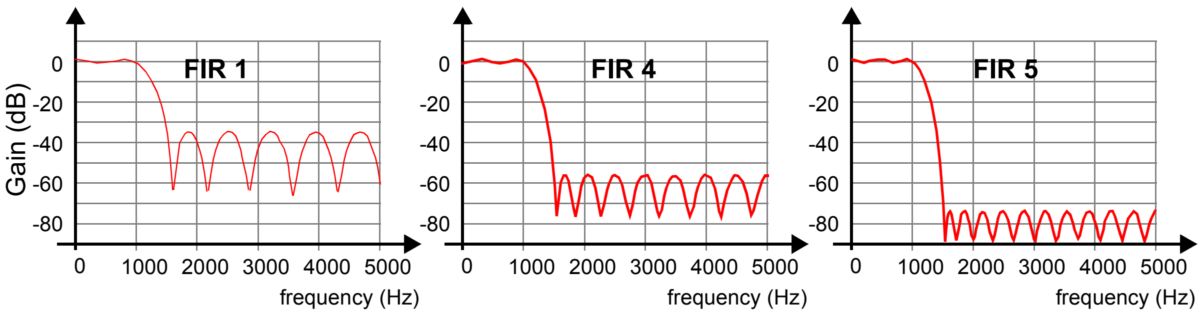
Low pass filters designed parameters: Stopband Attenuation = 40 dB; Passband Ripple = 1 dB and transition bandwidths, of 500, 200, and 50 Hz. The sharper the transition band the more filter weights that are required.

The respective impulse responses of FIR1, FIR2 and FIR3 are respectively 15, 69 and 269 weights long, with group delays of 7, 34 and 134 samples respectively.



The impulse responses of low pass filters FIR1, FIR2, and FIR3, all with 40 dB stopband attenuation, 1dB passband ripple, but transition bandwidths of 500, 200 and 50 Hz respectively. Clearly the more stringent the filter parameters, the longer the required impulse response.

Similarly if the stopband attenuation specification is increased, the number of filter weights required will again require to increase. For a low pass filter with a cut off frequency again at 1000 Hz, a transition bandwidth of 500 Hz and stopband attenuations of 40 dB, 60 dB and 80 dB:

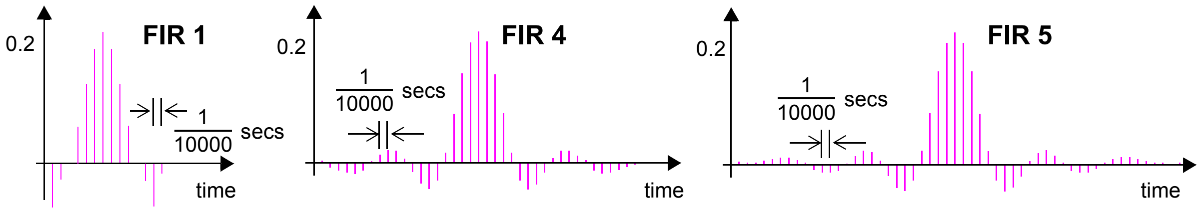


Transition Band: 1000 - 1500Hz
Stopband attenuation: 40 dB
No. of weights: 29

Transition Band: 1000 - 1200Hz
Stopband attenuation: 60 dB
No. of weights: 41

Transition Band: 1000 - 1100Hz
Stopband attenuation: 80 dB
No. of weights: 55

Low pass filters designed parameters: Transition Bandwidth = 500Hz; Passband Ripple = 1dB and stopband attenuations of 40 dB, 60 dB, and 80 dB.



The impulse responses of low pass filters FIR1, FIR4, and FIR5, all 1dB passband ripple, and transition bandwidths of 500 Hz and stopband attenuation of 40, 60 and 80dB respectively. Clearly the more stringent the filter parameters, the longer the required impulse response.

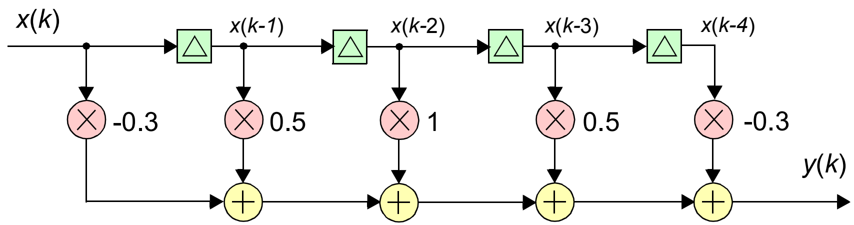
Similarly if the passband ripple parameter is reduced, then a longer impulse response will be required.

4.1 FIR Filter z-domain Zeroes

An important way of representing an FIR digital filter is with a z-domain plot of the filter zeroes. By writing the transfer function of an FIR filter in the z-domain, the resulting polynomial in *z* can be factorised to find the roots, which are in fact the “zeroes” of the digital filter. Consider a simple 5 weight FIR filter:

$$y(k) = -0.3x(k) + 0.5x(k-1) + x(k-2) + 0.5x(k-3) - 0.3x(k-4) \tag{64}$$

The signal flow graph of this filter can be represented as:



The signal flow graph for a 5 weight FIR filter.

The z-domain transfer function of this polynomial is therefore:

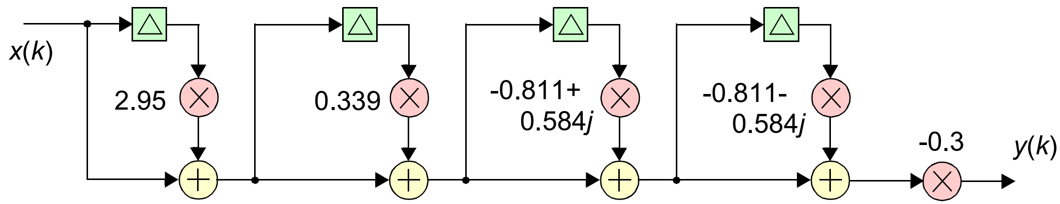
$$H(z) = \frac{Y(z)}{X(z)} = -0.3 + 0.5z^{-1} + z^{-2} + 0.5z^{-3} - 0.3z^{-4} \tag{65}$$

If the z-polynomial of Eq. 65 is factorised (using DSP design software rather than with paper and pencil!) then this gives for this example:

$$H(z) = -0.3(1 - 2.95z^{-1})(1 - (-0.811 + 0.584j)z^{-1})(1 - (-0.811 + 0.584j)z^{-1})(1 - 0.339z^{-1}) \tag{66}$$

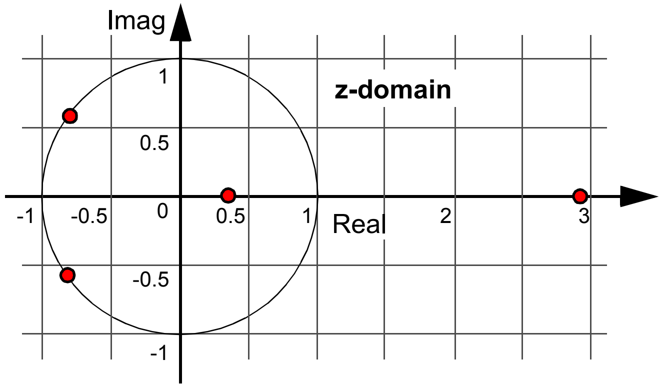
and the zeroes of the FIR filter (corresponding to the roots of the polynomial are, $z = 2.95, 0.339, -0.811 + 0.584j$, and $-0.811 - 0.584j$. (Note all quantities have been rounded to

3 decimal places). The corresponding SFG of the FIR filter written in the zero form of Eq. 66 is therefore:



The signal flow graph of four first order cascaded filters corresponding to the same impulse response as the 5 weight filter shown above. The first order filter coefficients correspond to the zeroes of the 5 weight filter.

The zeroes of the FIR filter can also be plotted on the z-domain plane:



The zeroes of the FIR filter in Eq. 65. Note that some of roots are complex. In the case of an FIR filter with real coefficients the zeroes are always symmetric about the x-axis (conjugate pairs) such that when the factorised polynomial is multiplied out there are no imaginary values.

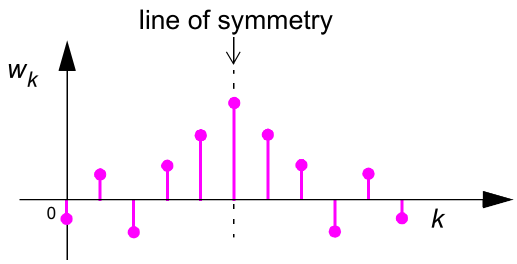
If all of the zeroes of the FIR filter are within the unit circle then the filter is said to be minimum phase.

4.2 FIR Linear Phase

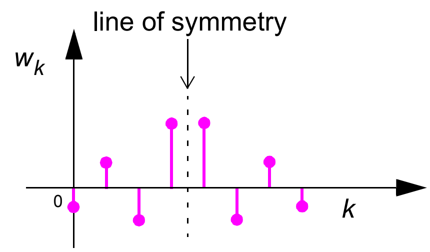
If the weights of an N weight real valued FIR filter are symmetric or anti-symmetric, i.e.

$$: \quad w(n) = \pm w(N-1-n) \tag{67}$$

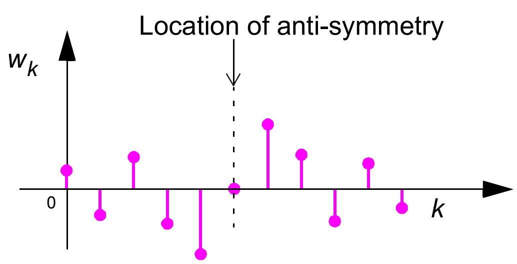
then the filter has linear phase. This means that all frequencies passing through the filter are delayed by the same amount. The impulse response of a linear phase FIR filter can have either an even or odd number of weights.



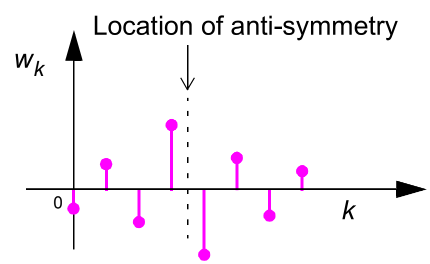
Symmetric impulse response of an 11 (odd number) weight linear phase FIR filter.



Symmetric impulse response of an 8 (even number) weight linear phase FIR filter.



Anti-symmetric impulse response of an 11 (odd number) weight linear phase FIR filter.



Anti-symmetric impulse response of an 8 (even number) weight linear phase FIR filter.

The z-domain plane pole zero plot of a linear phase filter will always have conjugate pair zeroes, i.e. the zeroes are symmetric about the real axis:

The desirable property of linear phase is particularly important in applications where the phase of a signal carries important information. To illustrate the linear phase response, consider inputting a cosine wave of frequency f, sampled at f_s samples per second (i.e. $\cos 2\pi fk/f_s$) to a symmetric impulse response FIR filter with an even number of weights N (i.e. $w_n = w_{N-n}$ for $n = 0, 1, \ldots, N/2-1$). For notational convenience let $\omega = 2\pi f/f_s$:

$$
\begin{aligned}
y(k) &= \sum_{n=0}^{N-1} w_n \cos\omega(k-n) = \sum_{n=0}^{N/2-1} w_n(\cos\omega(k-n) + \cos\omega(k-N+n)) \\
&= \sum_{n=0}^{N/2-1} 2w_n \cos\omega(k-N/2)\cos\omega(n-N/2) \\
&= 2\cos\omega(k-N/2) \sum_{n=0}^{N/2-1} w_n \cos\omega(n-N/2) \\
&= M \cdot \cos\omega(k-N/2), \qquad \text{where } M = \sum_{n=0}^{N/2-1} 2w_n \cos\omega(n-N/2)
\end{aligned}
\tag{68}
$$

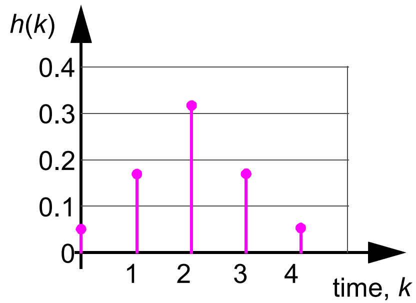
where the trigonometric identity, $\cos A + \cos B = 2\cos((A+B)/2)\cos((A-B)/2)$ has been used. From this equation it can be seen that regardless of the input frequency, the input cosine wave is delayed only by $N/2$ samples, often referred to as the group delay, and its magnitude is scaled by the factor M. Hence the phase response of such an FIR is simply a linear plot of the straight line defined by $\omega N/2$. Group delay is often defined as the differentiation of the phase response with respect to angular frequency. Hence, a filter that provides linear phase has a group delay that is constant for all frequencies. An all-pass filter with constant group delay (i.e., linear phase) produces a pure delay for any input time waveform.

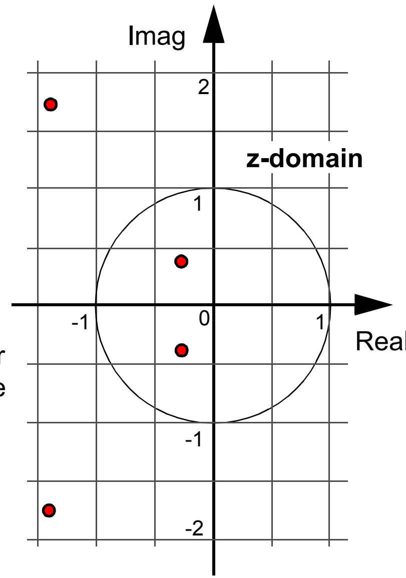
Linear phase FIR filters can be implemented with $N/2$ multiplies and N accumulates compared to the N MACs required by an FIR filter with a non-symmetric impulse response. This can be illustrated by rewriting the output of a symmetric FIR filter with an even number of coefficients:

$$y(k) = \sum_{n=0}^{N-1} w_n x(k-n) = \sum_{n=0}^{N/2-1} w_n[x(k-n) + x(k-N+n)] \tag{69}$$

Although the number of multiplies is halved, most DSP processors can perform a multiply-accumulate in the same time as an addition so there is not necessarily a computational advantage for the implementation of a symmetric FIR filter on a DSP device. One drawback of a linear phase filter is of course that they always introduce a delay.

Linear phase FIR filters are non-minimum phase, i.e. they will always have zeroes which are outside the unit circle. For the z-domain plane plot of the z-transform of a linear phase filter, for all zeroes that are not on the unit circle, there will be a complex conjugate reciprocal of that zero. For example:





The impulse response of a simple 5 weight linear phase FIR filter and the corresponding z-domain plane plot. Note that for the zeroes inside the unit circle at $z = -0.286 \pm 0.3526j$, there are conjugate reciprocal zeroes at:

$$z = \frac{1}{-0.286 \pm 0.3526j} = -1.384 \pm 1.727j$$

4.3 Minimum/Maximum Phase

If the zeroes of an FIR filter all lie within the unit circle on the z-domain plane, then the filter is said to be minimum phase. If the zeroes all lie outside then the filter is maximum phase. If there are zeroes both inside and outside the filter then the filter is non-minimum phase. One simple property is that the inverse filter of a minimum phase FIR filter is a stable IIR filter, i.e. all of the poles lie within the unit circle.

4.4 Order Reversed FIR Filter

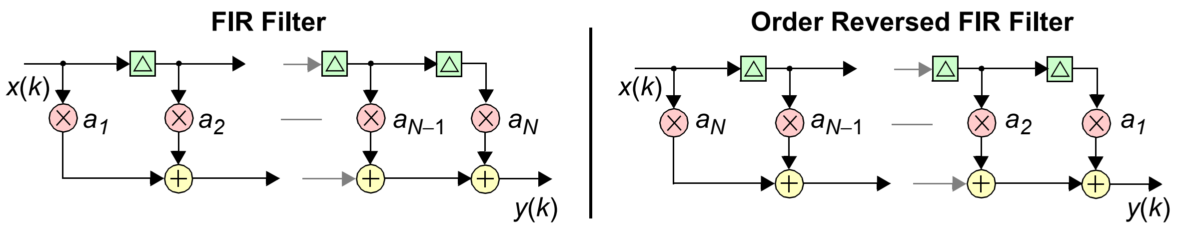
Consider the general finite impulse response filter with transfer function denoted as $H(z)$:

$$H(z) = a_1 + a_2 z^{-1} + \ldots + a_{N-1} z^{-N+1} + a_N z^{-N} \tag{70}$$

The order reversed FIR filter transfer function, $H_r(z)$ is given by:

$$H_r(z) = a_N + a_{N-1} z^{-1} + \ldots + a_2 z^{-N+1} + a_1 z^{-N} \tag{71}$$

The respective FIR filter signal flow graphs (SFG) are simply:



The signal flow graph for an $N+1$ weight FIR filter and the order reversed FIR filter. The order reversed FIR filter is same order as the original FIR filter but with the filter weights in opposite order.

From the z-domain functions above it is easy to show that $H_r(z) = z^{-N} H(z^{-1})$. The order reversed FIR filter has exactly the same magnitude frequency response as the original FIR filter:

$$
\begin{aligned}
|H_r(z)|_{z=e^{j\omega}} &= |z^{-N}H(z^{-1})|_{z=e^{j\omega}} = |e^{-j\omega N}H(e^{-j\omega})| = |H(e^{-j\omega})| = |H(e^{j\omega})| \\
&= |H(z)|_{z=e^{j\omega}}
\end{aligned}
\tag{72}
$$

The phase response of the two filters are, however, different. The difference to the phase response can be noted by considering that the zeroes of the order reversed FIR filter are the inverse of the zeroes of the original FIR filter, i.e. if the zeroes of Eq. 71 are $\alpha_1, \alpha_2, \ldots \alpha_{N-1}, \alpha_N$:

$$
H(z) = (1-\alpha_1 z^{-1})(1-\alpha_2 z^{-1})\ldots(1-\alpha_{N-1}z^{-1})(1-\alpha_N z^{-1})
\tag{73}
$$

then the zeroes of the order reversed polynomial are $\alpha_1^{-1}, \alpha_2^{-2}, \ldots \alpha_{N-1}^{-1}, \alpha_N^{-1}$ which can be seen from:

$$
\begin{aligned}
H_r(z) &= z^{-N}H(z^{-1}) \\
&= z^{-N}(1-\alpha_1 z)(1-\alpha_2 z)\ldots(1-\alpha_{N-1}z)(1-\alpha_N z) \\
&= (z^{-1}-\alpha_1)(z^{-1}-\alpha_2)\ldots(z^{-1}-\alpha_{N-1})(z^{-1}-\alpha_N) \\
&= \frac{(-1)^N}{\alpha_1\alpha_2\ldots\alpha_{N-1}\alpha_N}(1-\alpha_1^{-1}z^{-1})(1-\alpha_2^{-1}z^{-1})\ldots(1-\alpha_{N-1}^{-1}z^{-1})(1-\alpha_N^{-1}z^{-1})
\end{aligned}
\tag{74}
$$

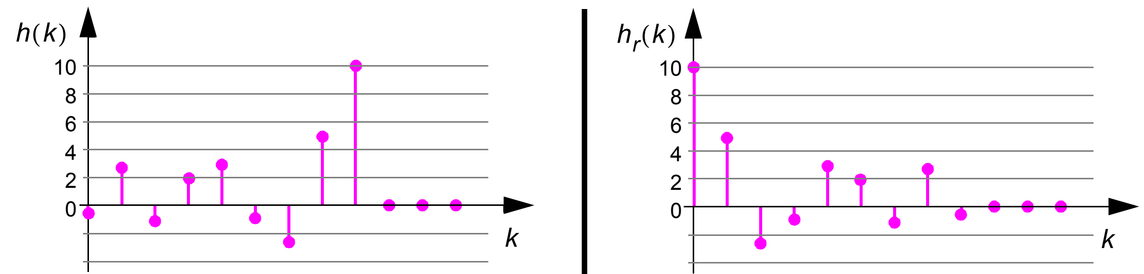
As examples consider the 8 weight FIR filter

$$
H(z) = 10 + 5z^{-1} - 3z^{-2} - z^{-3} + 3z^{-4} + 2z^{-5} - z^{-6} + 0.5z^{-7}
\tag{75}
$$

and the corresponding order reversed FIR filter:

$$
H(z) = 0.5 - z^{-1} + 2z^{-2} + 3z^{-3} - z^{-4} - 3z^{-5} + 5z^{-6} + 10z^{-7}
\tag{76}
$$

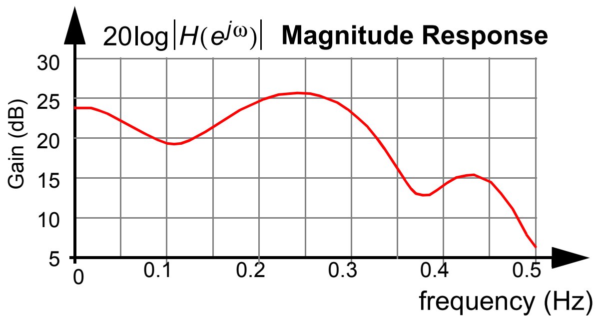
Assuming a sampling frequency of $f_s = 1$, the impulse response of both filters are easily plotted as:

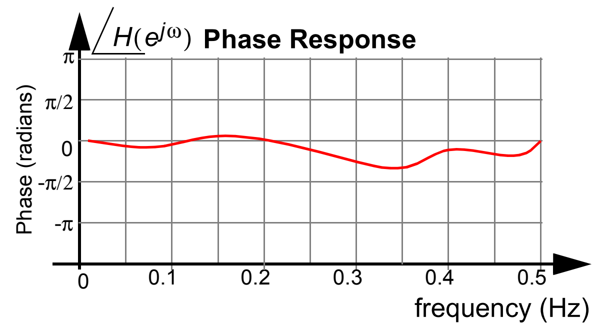


Impulse response, $h(k)$ of simple FIR filter

Order reversed impulse response, $h_r(k)$.

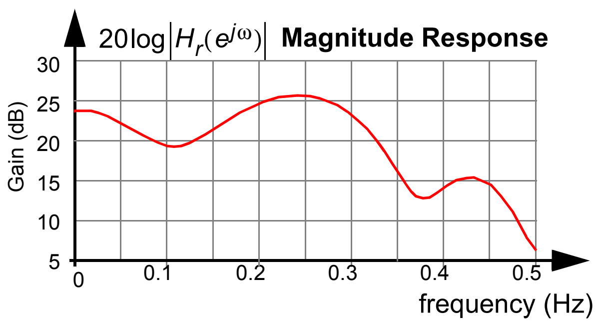
The corresponding magnitude and phase frequency responses of both filters are:

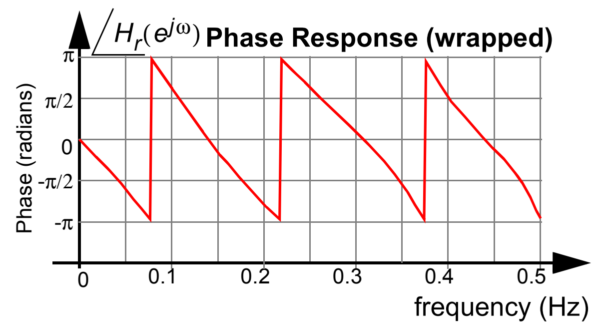




Magnitude and phase frequency response of FIR filter

$$H(z) = 10 + 5z^{-1} - 3z^{-2} - z^{-3} + 3z^{-4} + 2z^{-5} - z^{-6} + 0.5z^{-7} \quad (77)$$





Magnitude and phase frequency response of order reversed FIR filter

$$H_r(z) = 0.5 - z^{-1} + 2z^{-2} + 3z^{-3} - z^{-4} - 3z^{-5} + 5z^{-6} + 10z^{-7} \quad (78)$$

and the z-domain plots of both filter zeroes are:

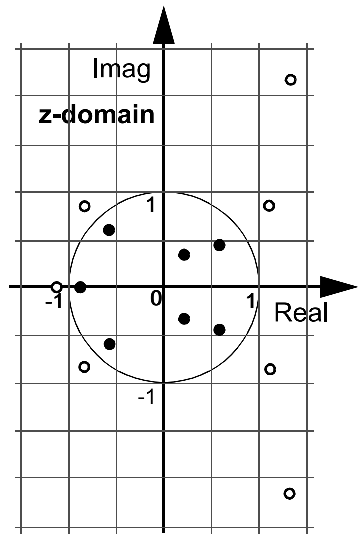
● **- Zeroes of FIR filter *H*(*z*)**

○ **- Zeroes of order reversed FIR filter *H*$_r$(*z*)**

For a zero $\alpha = x + jy$ we note that $|\alpha| = \sqrt{x^2 + y^2}$ and therefore for related the order reversed filter zero at $1/\alpha$ we note:

$$\left|\frac{1}{\alpha}\right| = \left|\frac{1}{x + jy}\right| = \frac{|x - jy|}{x^2 + y^2} = \frac{x^2 + y^2}{\sqrt{x^2 + y^2}} = \frac{1}{\sqrt{x^2 + y^2}} \qquad (79)$$

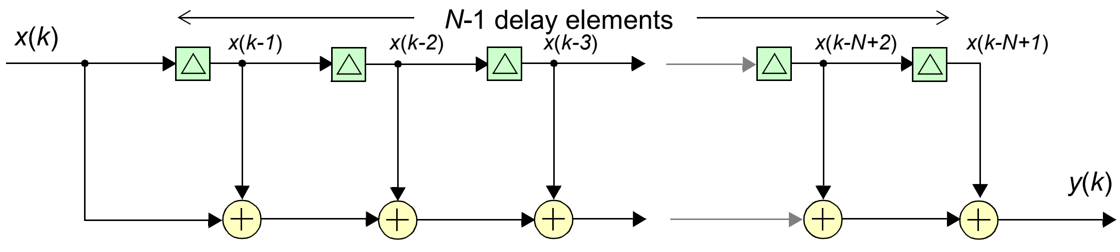
For this particular example $H(z)$ is clearly minimum phase (all zeroes inside the unit circle), and therefore $H_r(z)$ is maximum phase (all zeroes outside of the unit circle.



4.5 Moving Average (MA) FIR Filter

The moving average (MA) filter “usually” refers to an FIR filter of length N where all filter weights have the value of 1. (The term MA is however sometimes used to mean any (non-recursive) FIR filter usually within the context of stochastic signal modelling).

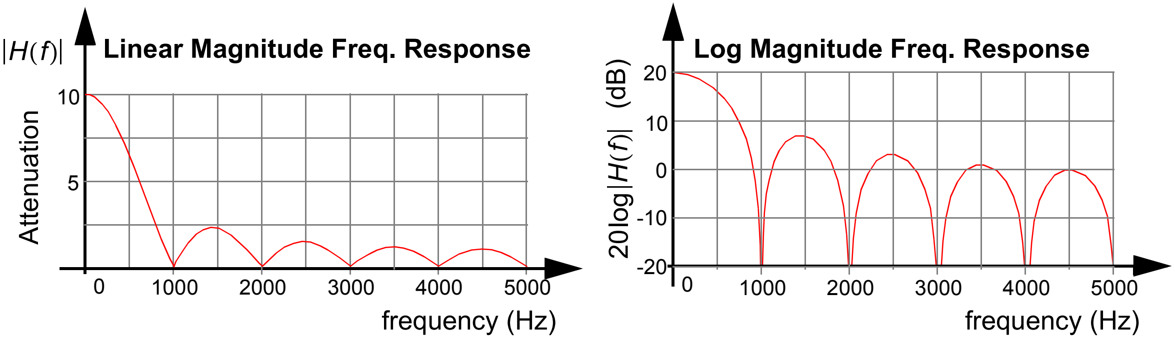
The moving average filter is a very simple form of low pass filter often found in applications where computational requirements need to be kept to a minimum. A moving average filter produces an output sample at time k by adding together the last N input samples (including the current one). This can be represented on a simple signal flow graph and with discrete equations as:



$$y(k) = x(k) + x(k-1) + x(k-2) + x(k-3) + \ldots\ldots + x(k-N+1) = \sum_{n=0}^{N-1} x(k-n)$$

The signal flow graph and output equation for a moving average FIR filter. The moving average filter requires no multiplications, only N additions.

As an example the magnitude frequency domain representations of a moving average filter with 10 weights is:



The linear and logarithmic frequency responses of a 10 weight moving average FIR filter. The peak of the first sidelobe of any moving average filter is always approximately 13 dB below the gain at 0 Hz.

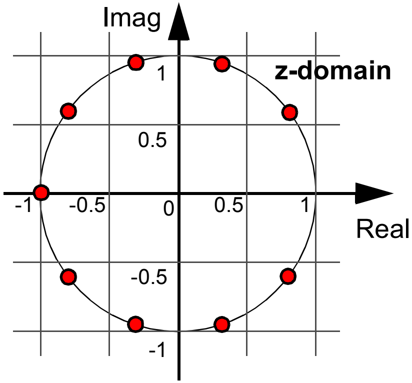
In terms of the z-domain, we can write the transfer function of the moving average FIR filter as:

$$H(z) = \frac{Y(z)}{X(z)} = 1 + z^{-1} + z^{-2} + \ldots + z^{-N+1}$$
$$= \sum_{i=0}^{N-1} z^{-i} = \frac{1 - z^{-N}}{1 - z^{-1}} \tag{80}$$

recalling that the sum of a geometric series $\{1, r, r^2, \ldots, r^m\}$ is given by $(1 - r^{m+1})/(1 - r)$. If the above moving average transfer function polynomial is factorized, this therefore represents a transfer function with N zeroes and a single pole at $z = 1$, which is of course cancelled out by a zero at $z = 1$ since an FIR filter has no poles associated with it. We can find the zeroes of the polynomial in Eq. 80 by solving:

$$
\begin{aligned}
1 - z^{-N} &= 0 \\
\Rightarrow z_n &= \sqrt[N]{1} \quad \text{where } n = 0 \ldots N - \\
\Rightarrow z_n &= \sqrt[N]{e^{j2\pi n}} \quad \text{noting } e^{j2\pi n} = \\
\Rightarrow z_n &= e^{\frac{j2\pi n}{N}}
\end{aligned}
\tag{81}
$$

which represents N zeroes equally spaced around the unit circle starting at $z = 1$, but with the $z = 1$ zero cancelled out by the pole at $z = 1$. The pole-zero z-domain plot for the above 10 weight moving average FIR filter is:



$$H(z) = \frac{Y(z)}{X(z)} = 1 + z^{-1} + z^{-2} + \ldots + z^{-9}$$
$$= \sum_{i=0}^{9} z^{-i} = \frac{1 - z^{-10}}{1 - z^{-1}}$$

The pole-zero plot for a moving average filter of length 10. As expected the filter has 9 zeroes equally spaced around the unit circle (save the one not present at $z = 1$). In some representations a pole and a zero may be shown at $z = 1$, however these cancel each other out. The use of a pole is only to simplify the z-transform polynomial expression.

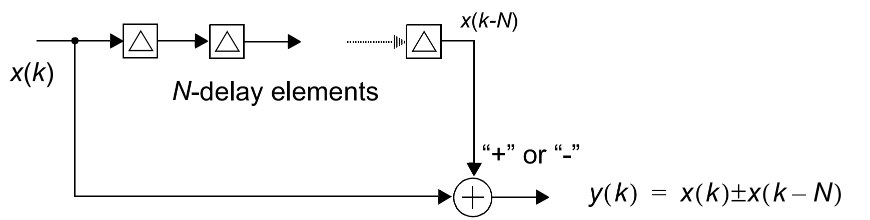
In general if a moving average filter has N weights then the width of the first (half) lobe of the mainlobe is $f_s/2N$ Hz, which is also the bandwidth of all of the sidelobes up to $f_s/2$.

The moving average filter shown will amplify an input signal by a factor of N. If no gain (or gain =1) is required at 0 Hz then the output of the filter should be divided by N. However one of the attractive features of a moving average filter is that it is simple to implement and the inclusion of a division is not conducive to this aim. Therefore should 0 dB be required at 0 Hz, then if the filter length is made a power of 2 (i.e. 8, 16, 32 and so on) then the division can be done with a simple shift right operation of the filter output, whereby each shift right divides by 2.

The moving average FIR filter is linear phase and has a group delay equal to half of the filter length ($N/2$).

4.6 Comb Filter

A comb digital filter is so called because the magnitude frequency response is periodic and resembles that of a comb. (It is worth noting that the term “comb filter” is not always used consistently in the DSP community). Comb filters are very simple to implement either as an FIR filter type structure where all weights are either 1, or 0, or as single pole IIR filters. Consider a simple FIR comb filter:



The simple comb filter can be viewed as an FIR filter where the first and last filter weights are 1, and all other weights are zero. The comb filter can be implemented with only a shift register, and an adder; multipliers are not required. If the two samples are added then the comb filter has a linear gain factor of 2 (i.e 6 dB) at 0 Hz (DC) thus in some sense giving a low pass characteristics at low frequencies. And if they are subtracted the filter has a gain of 0 giving in some sense a band stop filter characteristic at low frequencies.

The transfer function for the FIR comb filters can be found as:

$$
\begin{aligned}
Y(z) &= X(z) \pm z^{-N} X(z) = (1 \pm z^{-N}) X(z) \\
\Rightarrow H(z) &= \frac{Y(z)}{X(z)} = (1 \pm z^{-N})
\end{aligned}
\tag{82}
$$

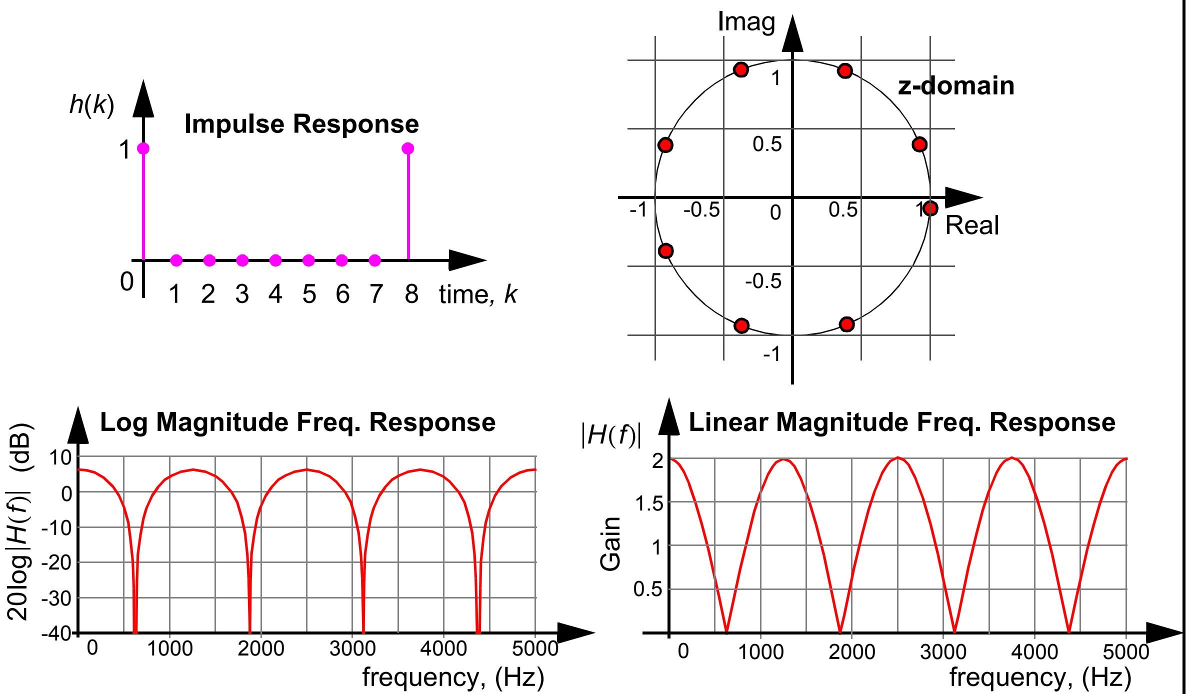
The zeroes of the comb filter, are the N roots of the z-domain polynomial $1 \pm z^{-N}$: Therefore for the case where the samples are subtracted:

$$
\begin{aligned}
1 - z^{-N} &= 0 \\
\Rightarrow z_n &= \sqrt[N]{1} \quad \text{where } n = 0 \ldots N-1 \\
\Rightarrow z_n &= \sqrt[N]{e^{j2\pi n}} \quad \text{noting } e^{j2\pi n} = 1 \\
\Rightarrow z_n &= e^{\frac{j2\pi n}{N}}
\end{aligned}
\tag{83}
$$

And for the case where the samples are added:

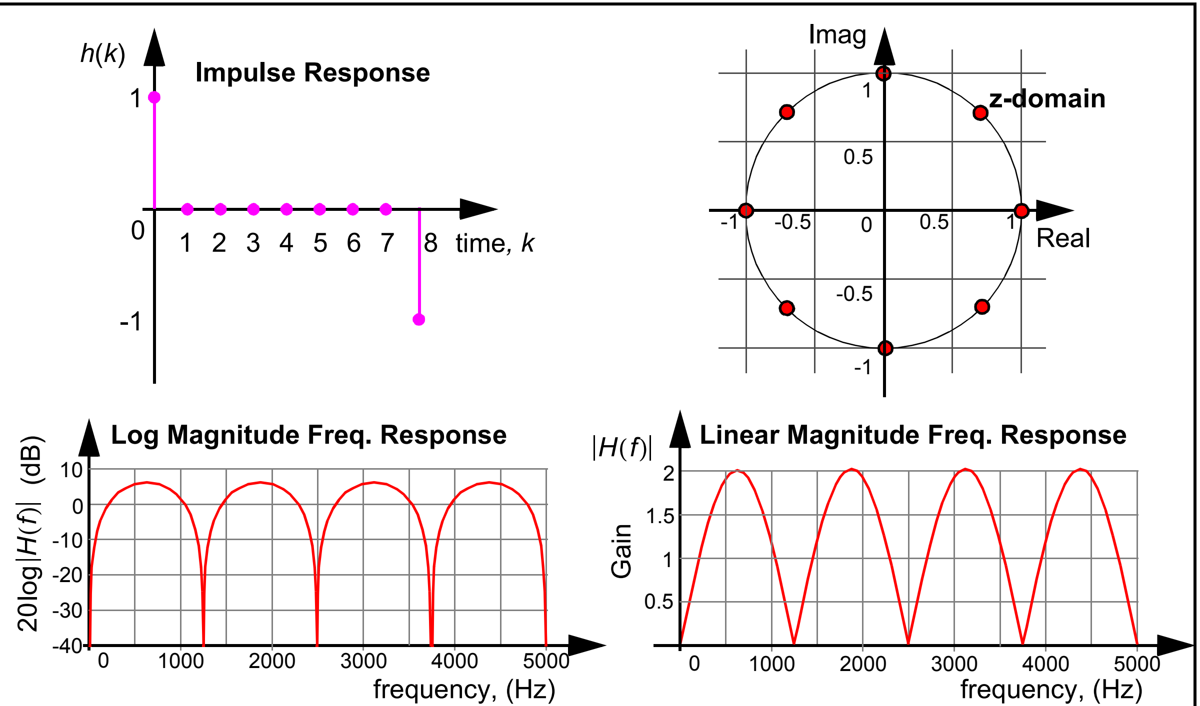
$$
\begin{aligned}
1 + z^{-N} &= 0 \\
\Rightarrow z_n &= \sqrt[N]{-1} \quad \text{where } n = 0 \ldots N-1 \\
\Rightarrow z_n &= \sqrt[N]{e^{j2\pi\left(n + \frac{1}{2}\right)}} \quad \text{noting } e^{j2\pi\left(n + \frac{1}{2}\right)} = -1 \\
\Rightarrow z_n &= e^{\frac{j2\pi\left(n + \frac{1}{2}\right)}{N}}
\end{aligned}
\tag{84}
$$

As an example, consider a comb filter $H(z) = 1 + z^{-8}$ and a sampling rate of $f_s = 10000$ Hz. The impulse response, $h(k)$, frequency response, $H(f)$, and zeroes of the filter can be illustrated as:



The impulse response, z-domain plot of the zeroes, and magnitude frequency response of the comb filter, $H(z) = 1 + z^{-8}$. Note that the comb filter is like a set of frequency selective bandpass filters, with the first half-band filter having a low pass characteristic. The number of bands from 0 Hz to $f_s/2$ is $N/2$. The zeroes are spaced equally around the unit circle and symmetrically about the x-axis with no zero at $z = 1$. (There is a zero at $z = -1$ if N is odd.)

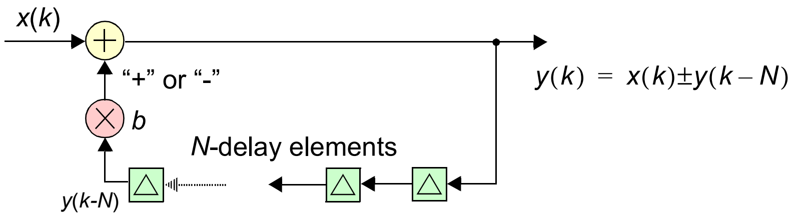
For the comb filter $H(z) = 1 - z^{-8}$ and a sampling rate of $f_s = 10000$ Hz. The impulse response, $h(k)$, frequency response, $H(f)$, and zeroes of the filter are:



The impulse response, z-domain plot of the zeroes, and magnitude frequency response of the comb filter, $H(z) = 1 - z^{-8}$. The zeroes are spaced equally around the unit circle and symmetrically about the x-axis. There is a zero at $z = 1$.There is not a zero a $z = -1$ if N is odd.

FIR comb filters have linear phase and are unconditionally stable (as are all FIR filters).

Another type of comb filter magnitude frequency response can be produced from a single pole IIR filter:

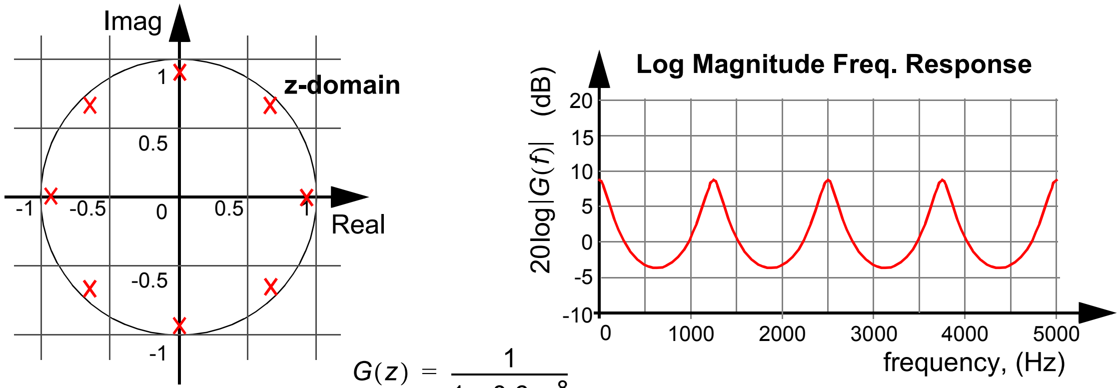


A single pole IIR comb filter. The closer the weight value b is to 1, then the sharper the teeth of the *comb filter* in the frequency domain (see below). b is of course less than 1, or instability results.

Unlike the FIR comb filter note that this comb filter does require at least one multiplication operation. Consider the difference equation of the above single pole IIR *comb filter:*

$$
\begin{aligned}
y(k) &= x(k) \pm b y(k-N) \\
\Rightarrow G(z) &= \frac{Y(z)}{X(z)} = \frac{1}{1 \pm b z^{-N}}
\end{aligned}
\qquad (85)
$$

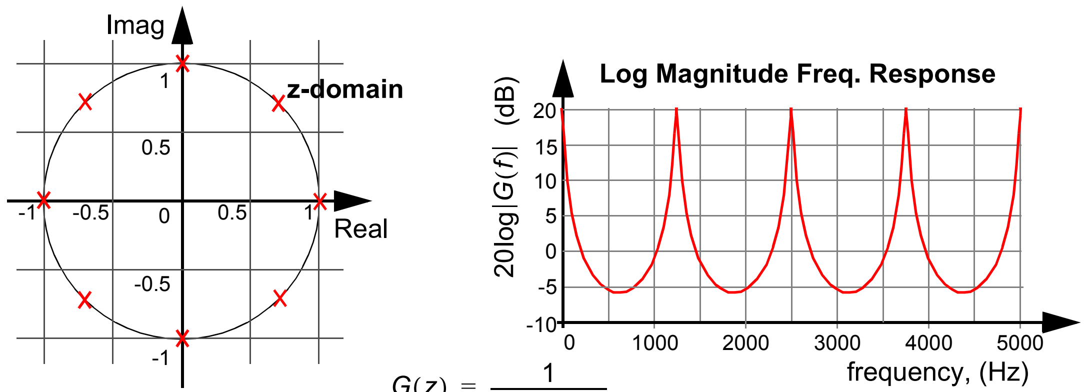
For a sampling rate of $f_s = 10000\text{ Hz}$, $N = 8$ and $b = 0.6$ the impulse response $g(n)$, the frequency response, $G(f)$, and poles of the filter are:



$$G(z) = \frac{1}{1 - 0.6z^{-8}}$$

The z-domain plot of the filter poles and magnitude frequency response of one pole comb filter. The poles are inside the unit circle and lie on a circle of radius $0.6^{1/8} = 0.938\ldots$. As the feedback weight value, b, is decreased (closer to 0), then the poles move away from the unit circle towards the origin, and the peaks of the magnitude frequency response become less sharp and provide less gain.

Increasing the feedback weight, b, to be very close to 1, the "teeth" of the filter become sharper and the gain increases:



$$G(z) = \frac{1}{1 - 0.9z^{-8}}$$

The z-domain plot of the filter poles and magnitude frequency response of a one pole comb filter. The poles are just inside the unit circle and lie on a circle of radius $0.9^{1/8} = 0.987\ldots$.

Of course if b is increased such that $b \geq 1$ then the filter is unstable.

It is worth noting again that the term "*comb filter*" is used by some to refer to the single pole IIR comb filter described above, and the term "inverse *comb filter*" to the FIR *comb filter* both described above. Other authors refer to both as comb filters. The uniting feature however of all comb filters is the periodic (comb like) magnitude frequency response.

4.7 Differentiator

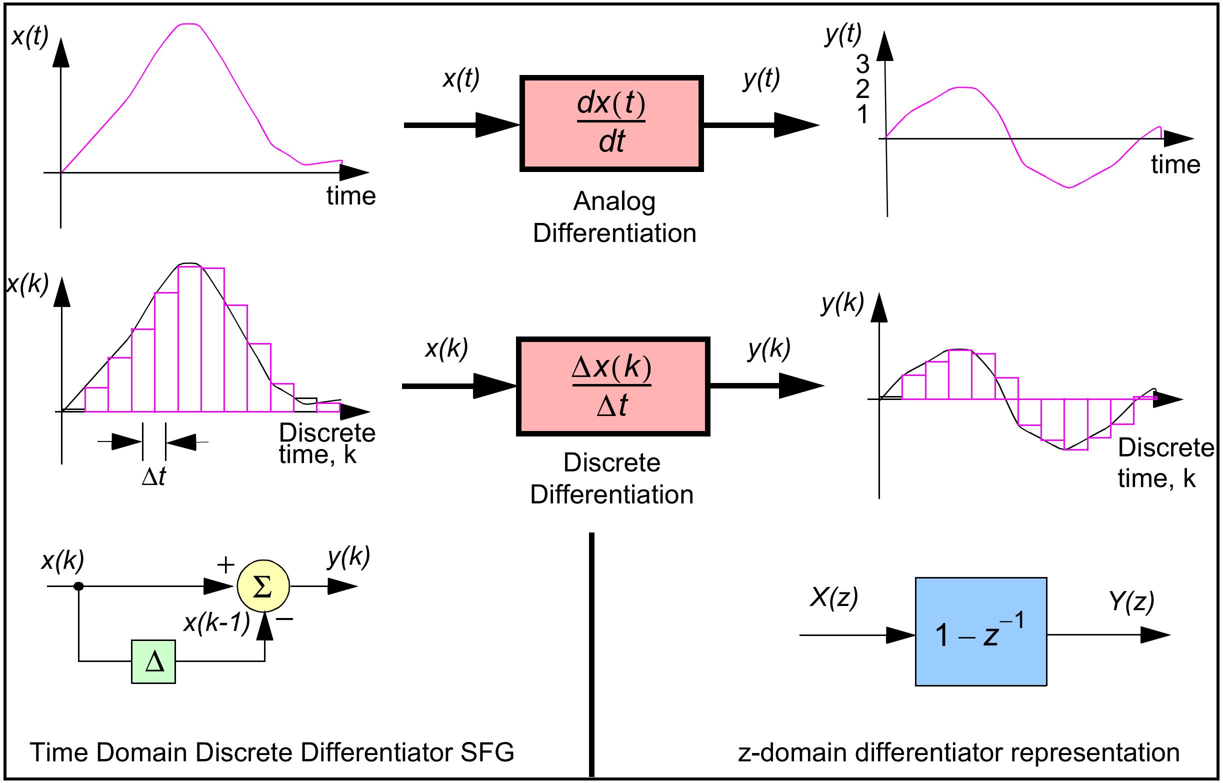
A *differentiator* is a two weight filter that is implemented using a digital delay element, and a summing element to calculate:

$$y(k) = x(k) - x(k-1) \tag{86}$$

In the z-domain the transfer function of a *differentiator* is:

$$\begin{aligned} Y(z) &= X(z) - z^{-1}X(z) \\ \Rightarrow \quad \frac{Y(z)}{X(z)} &= 1 - z^{-1} \end{aligned} \tag{87}$$

When viewed in the frequency domain a *differentiator* has the characteristics of a high pass filter. Thus differentiating a signal with additive noise tends to emphasize or enhance the high frequency components of the additive noise.

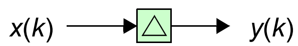


4.8 All-pass FIR Filter

An *all-pass filter* passes all input frequencies with the same gain, although the phase of the signal will be modified. (A true *all-pass filter* has a gain of one.) *All-pass filters* are used for applications such as group delay equalisation, notch filtering design, Hilbert transform implementation, musical instruments synthesis.

The simplest all pass filter is a simple delay! This "filter" passes all frequencies with the same gain, has linear phase response and introduces a group delay of one sample at all frequencies:

time domain



$$y(k) = x(k-1)$$

z-domain

$$Y(z) = z^{-1}H(z)$$

$$H(z) = \frac{Y(z)}{X(z)} = z^{-1}$$

A simple *all pass filter*. All frequencies are passed with the same gain.

A more general representation of some types of *all pass filters* can be represented by the general z-domain transfer function for an infinite impulse response (IIR) N pole, N zero filter:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{a_0^* z^{-N} + a_1^* z^{-N+1} + \ldots + a_{N-1}^* z^{-1} + a_N^*}{a_0 + a_1 z^{-1} + \ldots + a_{N-1} z^{-N+1} + a_N z^{-N}} = \frac{z^{-N} A^*(z^{-1})}{A(z)} \qquad (88)$$

where a^* is the complex conjugate of a. Usually the filter weights are real, therefore $a = a^*$, and we set $a_0 = 1$:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{z^{-N} + a_1 z^{-N+1} + \ldots + a_{N-1} z^{-1} + a_N}{1 + a_1 z^{-1} + \ldots + a_{N-1} z^{-N+1} + a_N z^{-N}} = \frac{z^{-N} A(z^{-1})}{A(z)} \qquad (89)$$

We can easily show that $|H(z)| = a_N$ (see below) for all frequencies. Note that the numerator polynomial $z^{-N}A(z)$ is simply the ordered reversed z-polynomial of the denominator $A(z)$. For an input signal $x(k)$ the discrete time output of an *all-pass filter* is:

$$\begin{aligned} y(k) = {} & a_N x(k) + a_{N-1} x(k-1) + \ldots + a_1 x(k-N+1) + x(k-N) + \\ & + a_1 y(k-1) + \ldots + a_{N-1} y(k+N-1) + a_N x(k-N) \end{aligned} \tag{90}$$

In order to be stable, the poles of the *all-pass filter* must lie within the unit circle. Therefore, for the denominator polynomial, if the N roots of the polynomial $A(z)$ are:

$$A(z) = (1 - p_1 z^{-1})(1 - p_2 z^{-1})\ldots(1 - p_N z^{-1}) \tag{91}$$

then $|p_n| < 1$ for $n = 1$ to N in order to ensure all poles are within the unit circle. The poles and zeroes of the all pass filter are therefore:

$$H(z) = \frac{a_N(1 - p_1^{-1} z^{-1})(1 - p_2^{-1} z^{-1})\ldots(1 - p_N^{-1} z^{-1})}{(1 - p_1 z^{-1})(1 - p_2 z^{-1})\ldots(1 - p_N z^{-1})} \tag{92}$$

where the roots of the zeroes polynomial $A(z^{-1})$ are easily calculated to be the inverse of the poles (see following example).

To illustrate the relationship between roots of z-domain polynomial and of its order reversed polynomial, consider a polynomial of order 3 with roots at $z = p_1$ and $z = p_2$:

$$\begin{aligned} 1 + a_1 z^{-1} + a_2 z^{-2} + a_3 z^{-3} &= (1 - p_1 z^{-1})(1 - p_2 z^{-1})(1 - p_3 z^{-1}) \\ &= 1 - (p_1 + p_2 + p_3) z^{-1} + (p_1 p_2 + p_2 p_3 + p_1 p_3) z^{-2} + p_1 p_2 p_3 z^{-3} \end{aligned} \tag{93}$$

Then replacing z with z^{-1} gives:

$$1 + a_1 z^1 + a_2 z^2 + a_3 z^3 = (1 - p_1 z)(1 - p_2 z)(1 - p_3 z) \tag{94}$$

and therefore multiplying both sides by z^{-3} gives:

$$\begin{aligned} z^{-3}(1 + a_1 z^1 + a_2 z^2 + a_3 z^3) &= z^{-3}(1 - p_1 z)(1 - p_2 z)(1 - p_3 z) \\ z^{-3} + a_1 z^{-2} + a_2 z^{-1} + a_3 &= (z^{-1} - p_1)(z^{-1} - p_2)(z^{-1} - p_3) \\ &= -p_1 p_2 p_3 (1 - p_1^{-1} z^{-1})(1 - p_2^{-1} z^{-1})(1 - p_3^{-1} z^{-1}) \\ &= -a_3 (1 - p_1^{-1} z^{-1})(1 - p_2^{-1} z^{-1})(1 - p_3^{-1} z^{-1}) \end{aligned} \tag{95}$$

hence revealing the roots of the order reversed polynomial to be at $z = 1/p_1$, $z = 1/p_2$ and $z = 1/p_3$.

Of course, if all of the poles of Eq. 92 lie within the z-domain unit circle then all of the zeroes of the denominator of Eq. 92 will necessarily lie outside of the unit circle of the z-domain, i.e. when $|p_n| < 1$ for $n = 1$ to N then $|p_n^{-1}| > 1$ for $n = 1$ to N. Therefore an *all pass filter* is maximum phase.

The magnitude frequency response of the pole at $z = p_i$ and the zero at $z = p_i^{-1}$ is:

$$|H_i(e^{j\omega})| = \left.\frac{|1 - p_i^{-1}z^{-1}|}{|1 - p_i z^{-1}|}\right|_{z = e^{j\omega}} = \frac{1}{|p_i|} \tag{96}$$

If we let $p_i = x + jy$ then the frequency response is found by evaluating the transfer function at $z = e^{j\omega}$:

$$H_i(e^{j\omega}) = \frac{1 - p_i^{-1}e^{-j\omega}}{1 - p_i e^{-j\omega}} = \frac{1}{p_i}\left(\frac{p_i - e^{-j\omega}}{1 - p_i e^{-j\omega}}\right) = \frac{1}{p_i}G(e^{j\omega}) \tag{97}$$

where $|G(e^{j\omega})| = 1$. This can be shown by first considering that:

$$G(e^{j\omega}) = = \frac{x + jy - (\cos\omega - j\sin\omega)}{1 - (x + jy)(\cos\omega - j\sin\omega)} = \frac{(x - \cos\omega) + j(y - \sin\omega)}{1 - x\cos\omega - y\sin\omega + j(x\sin\omega - y\cos\omega)} \tag{98}$$

and therefore the (squared) magnitude frequency response of $G(e^{j\omega})$ is:

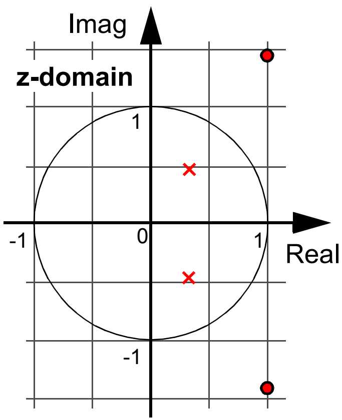
$$\begin{aligned}
|G(e^{j\omega})|^2 &= \frac{(x - \cos\omega)^2 + (y - \sin\omega)^2}{(1 - (x\cos\omega + y\sin\omega))^2 + (x\sin\omega - y\cos\omega)^2} \\
&= \frac{(x^2 - 2x\cos\omega + \cos^2\omega) + (y^2 - 2y\sin\omega + \sin^2\omega)}{1 - 2x\cos\omega - 2y\sin\omega + (x\cos\omega + y\sin\omega)^2 + x^2\sin^2\omega + y^2\cos^2\omega - 2xy\sin\omega\cos\omega} \\
&= \frac{(\sin^2\omega + \cos^2\omega) + x^2 + y^2 - 2x\cos\omega - 2b\sin\omega}{1 + x^2(\sin^2\omega + \cos^2\omega) + y^2(\sin^2\omega + \cos^2\omega) - 2x\cos\omega + 2y\sin\omega} \\
&= \frac{1 + x^2 + y^2 - 2x\cos\omega - 2y\sin\omega}{1 + x^2 + y^2 - 2x\cos\omega + 2y\sin\omega} = 1
\end{aligned} \tag{99}$$

Hence: $|H_i(e^{j\omega})| = \frac{1}{|p_i|} = \frac{1}{\sqrt{x^2 + y^2}}$

Therefore the magnitude frequency response of the all pass filter in Eq. 92 is indeed “flat” and given by:

$$|H(e^{j\omega})| = a_N|H_1(e^{j\omega})||H_2(e^{j\omega})|\ldots|H_N(e^{j\omega})| = \frac{a_N}{|p_1||p_2|\ldots|p_N|} = 1 \tag{100}$$

From Eq. 89 and 92 it is easy to show that $a_N = |p_1||p_2|\ldots|p_N|$.



Consider the poles and zeroes of a simple 2nd order all-pass filter transfer function (found by simply using the quadratic formula):

$$\begin{aligned} H(z) &= \frac{1+2z^{-1}+3z^{-2}}{3+2z^{-1}+z^{-2}} \\ &= \frac{(1-(1+j\sqrt{2})z^{-1})(1-(1-j\sqrt{2})z^{-1})}{3(1-(1/3+j\sqrt{2}/3)z^{-1})(1-(1/3-j\sqrt{2}/3)z^{-1})} \\ &= \frac{1}{|p_1||p_2|}\cdot\frac{(1-p_1^{-1}z^{-1})(1-p_2^{-1}z^{-1})}{(1-p_1z^{-1})(1-p_2z^{-1})} \end{aligned} \tag{101}$$

and obviously $p_1 = 1/3 - j\sqrt{2}/3$ and $p_2 = 1/3 + j\sqrt{2}/3$ and $p_1^{-1} = 1 - j\sqrt{2}$ and $p_2^{-1} = 1 + j\sqrt{2}$. This example demonstrates that given that the poles must be inside the unit circle for a stable filter, the zeroes will always be outside of the unit circle, i.e. maximum phase.

Any non-minimum phase system (i.e. zeroes outside the unit circle) can always be described as a cascade of a minimum phase filter and a maximum phase all-pass filter. Consider the non-minimum phase filter:

$$H(z) = \frac{(1-\alpha_1 z^{-1})(1-\alpha_2 z^{-1})(1-\alpha_3 z^{-1})(1-\alpha_4 z^{-1})}{(1-\beta_1 z^{-1})(1-\beta_2 z^{-1})(1-\beta_3 z^{-1})} \tag{102}$$

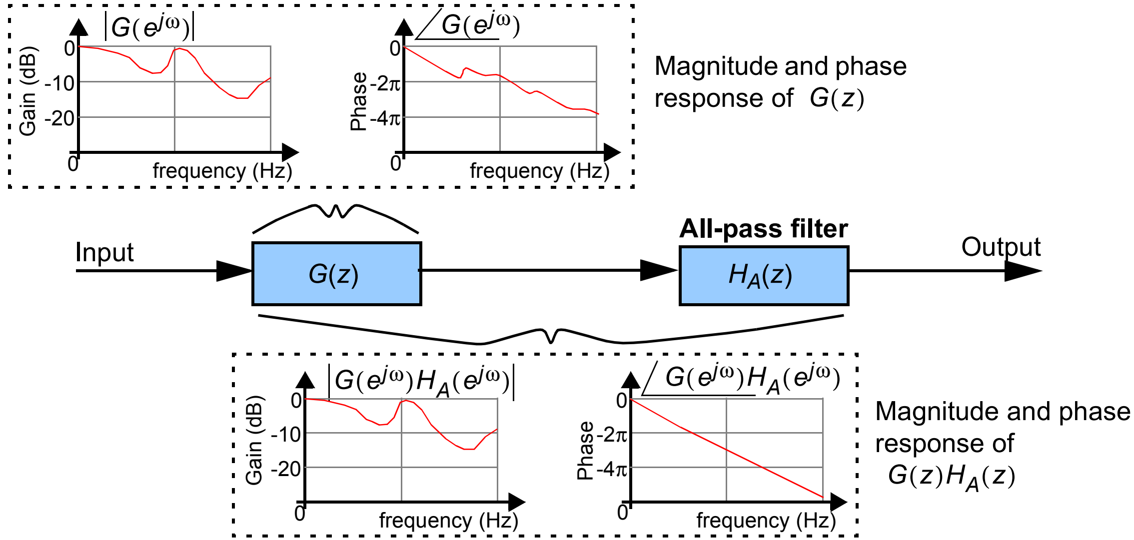
where the poles, β_1, β_2, and β_3 are inside the unit circle (to ensure a stable filter) and the zeroes α_1 and α_2 are inside the unit circle, but the zeroes α_3 and α_4 are outside of the unit circle. This filter can be written in the form of a minimum phase system cascaded with an all-pass filter by rewriting as:

$$\boxed{\begin{aligned} H(z) &= \left(\frac{(1-\alpha_1 z^{-1})(1-\alpha_2 z^{-1})(1-\alpha_3 z^{-1})(1-\alpha_4 z^{-1})}{(1-\beta_1 z^{-1})(1-\beta_2 z^{-1})(1-\beta_3 z^{-1})}\right)\left(\frac{(1-\alpha_3^{-1} z^{-1})(1-\alpha_4^{-1} z^{-1})}{(1-\alpha_3^{-1} z^{-1})(1-\alpha_4^{-1} z^{-1})}\right) \\ &= \underbrace{\left(\frac{(1-\alpha_1 z^{-1})(1-\alpha_2 z^{-1})(1-\alpha_3^{-1} z^{-1})(1-\alpha_4^{-1} z^{-1})}{(1-\beta_1 z^{-1})(1-\beta_2 z^{-1})(1-\beta_3 z^{-1})}\right)}_{\textbf{Minimum phase filter}}\underbrace{\left(\frac{((1-\alpha_3 z^{-1})(1-\alpha_4 z^{-1}))}{(1-\alpha_3^{-1} z^{-1})(1-\alpha_4^{-1} z^{-1})}\right)}_{\textbf{All-pass maximum phase filter}} \end{aligned}} \tag{103}$$

Therefore the minimum phase filter has zeroes inside the unit circle at $z = \alpha_3^{-1}$, $z = \alpha_4^{-1}$ and has exactly the same magnitude frequency response as the original filter and the gain of the all-pass filter being 1.

4.8.1 Phase Compensation

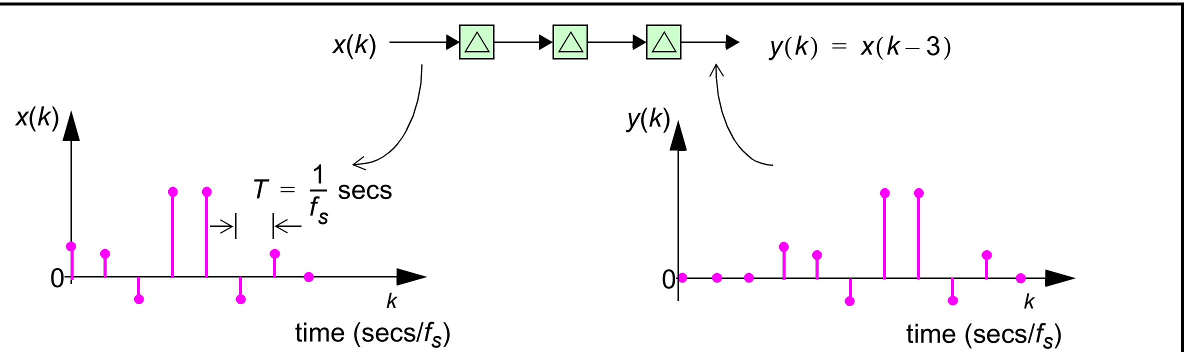
All pass filters are often used for phase compensation or group delay equalisation where the aim is to cascade an all-pass filter with a particular filter in order to achieve a linear phase response in the passband and leave the magnitude frequency response unchanged. (Given that signal information in the stopband is unwanted then there is usually no need to phase compensate there!). Therefore if a particular filter has a non-linear phase response and therefore non-constant group delay, then it may be possible to design a phase compensating *all-pass filter*:



Cascading an all pass filter $H_A(z)$ with a non-linear phase filter $G(z)$ in order to linearizethe phase response and therefore produce a constant group delay. The magnitude frequency response of the cascaded system is the same as the original system.

4.8.2 Fractional Delay Implementation

If it is required to delay a digital signal by a number of discrete sample delays this is easily accomplished using delay elements:



Delaying a signal by 3 samples, using simple delay elements.

Using DSP techniques to delay a signal by a time that is an integer number of sample delays $t_s = 1/f_s$ is therefore relatively straightforward. However, delaying by a time that is not an integer number of sampling delays (i.e a fractional delay) is less straightforward.

Another method uses a simple first order all pass filter, to "approximately" implement a fractional sampling delay. Consider the all-pass filter:

$$H(z) = \frac{z^{-1} + a}{1 + az^{-1}} \tag{104}$$

To find the phase response, we first calculate:

$$H(e^{j\omega}) = \frac{e^{-j\omega} + a}{1 + ae^{-j\omega}} = \frac{\cos\omega - j\sin\omega + a}{1 + a\cos\omega - ja\sin\omega} = \frac{(a + \cos\omega) - j\sin\omega}{1 + a\cos\omega - ja\sin\omega} \tag{105}$$

and therefore:

$$\angle H(e^{j\omega}) = \tan^{-1}\left(\frac{-\sin\omega}{a+\cos\omega}\right) + \tan^{-1}\left(\frac{a\sin\omega}{1+a\cos\omega}\right) \tag{106}$$

For small values of x the approximation $\tan^{-1}x \approx x$, $\cos x \approx 1$ and $\sin x \approx x$ hold. Therefore in Eq. 106, for small values of ω we get:

$$\angle H(e^{j\omega}) \approx \frac{-\omega}{a+1} + \frac{a\omega}{1+a} = -\frac{1-a}{1+a}\omega = \delta\omega \tag{107}$$

where $\delta = (1-a)/(1+a)$. Therefore at “small” frequencies the phase response is linear, thus giving a constant group delay of δ. Hence if a signal with a low frequency value f_i, where:

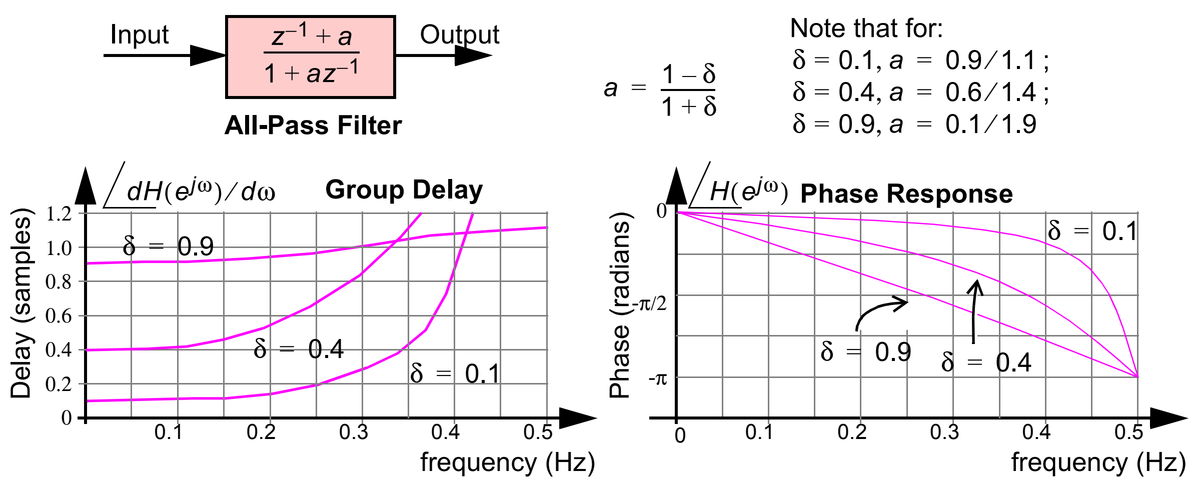
$$\frac{2\pi f_i}{f_s} << 1 \tag{108}$$

is required to be delayed by δ of a sample period ($t_s = 1/f_s$), then:

$$\Rightarrow \delta = \frac{1-a}{1+a} \qquad \Rightarrow a = \frac{1-\delta}{1+\delta} \tag{109}$$

Therefore for the sine wave input signal of $x(k) = \sin(2\pi f_i k/f_s)$ the output signal is approximately $y(k) \approx \sin((2\pi f_i(k-\delta))/f_s)$.

Parameters associated with creating delays of 0.1, 0.4, and 0.9 are shown below:

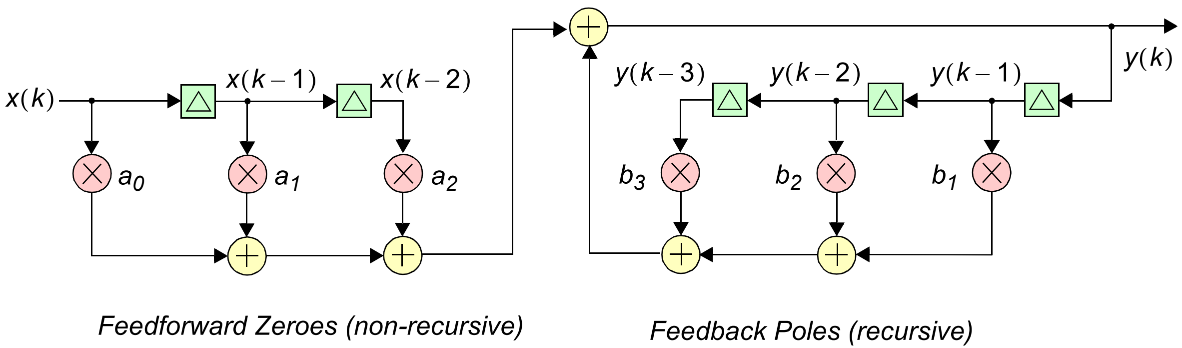


Phase response and group delay for a first order all pass filter implementing a fractional delay at low frequencies. For frequencies below $0.1f_s$ the phase response is "almost" linear, and therefore the group delay is effectively a constant. Note of course that for a stable filter, $a < 1$. The gain at all frequencies is 1 (a feature of all pass filters of course).

One area where fractional delays are useful is in musical instrument synthesis where accurate control of the feedback loop delay is desirable to allow accurate generation of musical notes with rich harmonics using "simple" filters. If a digital audio system is sampling at $f_s = 48000$ Hz then for frequencies up to around 4000 Hz very accurate control is available over the loop delay thus allowing accurate generation of musical note frequencies.

5 Infinite Impulse Response (IIR) Filter

An IIR filter is a digital filter which employs feedback to allow sharper frequency responses to be obtained for fewer filter coefficients. Unlike FIR filters, *IIR filters* can exhibit instability and must therefore be very carefully designed. The term infinite refers to the fact that the output from a unit pulse input will exhibit nonzero outputs for an arbitrarily long time. If the digital filter is IIR, then two weight vectors can be defined: one for the feedforward weights and one for the feedback weights:



$$y(k) = \sum_{n=0}^{2} a_n x(k-n) + \sum_{n=1}^{3} b_n y(k-n)$$

$$= a_0 x(k) + a_1 x(k-1) + a_2 x(k-2) + b_1 y(k-1) + b_2 y(k-2) + b_3 y(k-3)$$

$$\Rightarrow y(k) = \mathbf{a}^T \mathbf{x}(k) + \mathbf{b}^T \mathbf{y}(k-1) = \begin{bmatrix} a_0 & a_1 & a_2 \end{bmatrix} \begin{bmatrix} x(k) \\ x(k-1) \\ x(k-2) \end{bmatrix} + \begin{bmatrix} b_1 & b_2 & b_3 \end{bmatrix} \begin{bmatrix} y(k-1) \\ y(k-2) \\ y(k-3) \end{bmatrix}$$

A signal flow graph and equation for a 3 zero, 4 pole infinite impulse response filter.

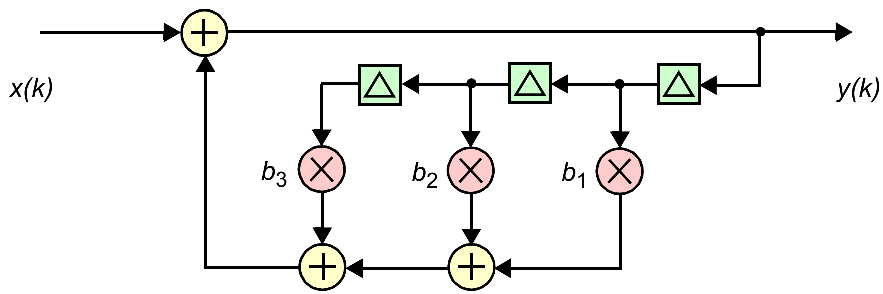
For the a 3 feedforward weight, and 3 feedback weight filter the z-domain transfer function is given as:

$$\frac{Y(z)}{X(z)} = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 - b_1 z^{-1} - b_2 z^{-2} - b_3 z^{-3}} = \frac{a_0(1-\alpha_1 z^{-1})(1-\alpha_2 z^{-1})}{(1-\beta_1 z^{-1})(1-\beta_2 z^{-1})(1-\beta_3 z^{-1})} = \frac{A(z)}{B(z)} \tag{110}$$

The roots of the numerator polynomial (i.e. $A(z) = 0$) give the zeroes of the filter and the roots of the denominator polynomial (i.e. $B(z) = 0$) give the poles of filter.

5.1 IIR Filter Stability

For stability of the IIR filter the magnitude of all poles must be less than 1. To see this requirement we can factorize the denominator of the IIR z-domain representation and view the signal flow graph in a more structured form. For an (all-pole) IIR filter the SFG is:



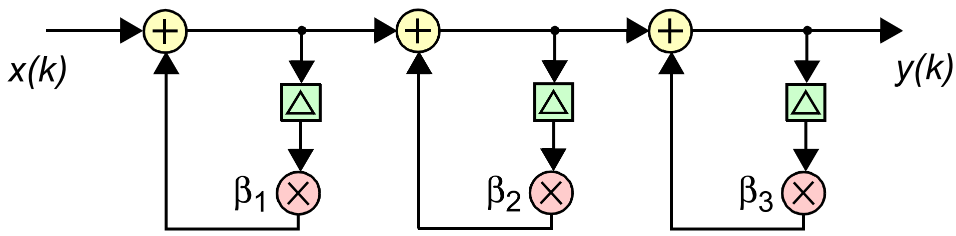
and the z-transform is therefore:

$$H(z) = \frac{1}{1 - b_1 z^{-1} - b_2 z^{-2} - b_3 z^{-3}}$$

Factorizing this equation gives:

$$H(z) = \frac{1}{(1-\beta_1 z^{-1})(1-\beta_2 z^{-1})(1-\beta_3 z^{-1})}$$

which can be redrawn in the form:



Stability requires that all $\{|\beta_1|, |\beta_2|, |\beta_3|\} < 1$, i.e. all roots (poles) of the polynomial are less than 1, otherwise if anyone of the simple first order filters has a coefficient greater than 1, then the entire cascade is unstable.

5.2 Integrator

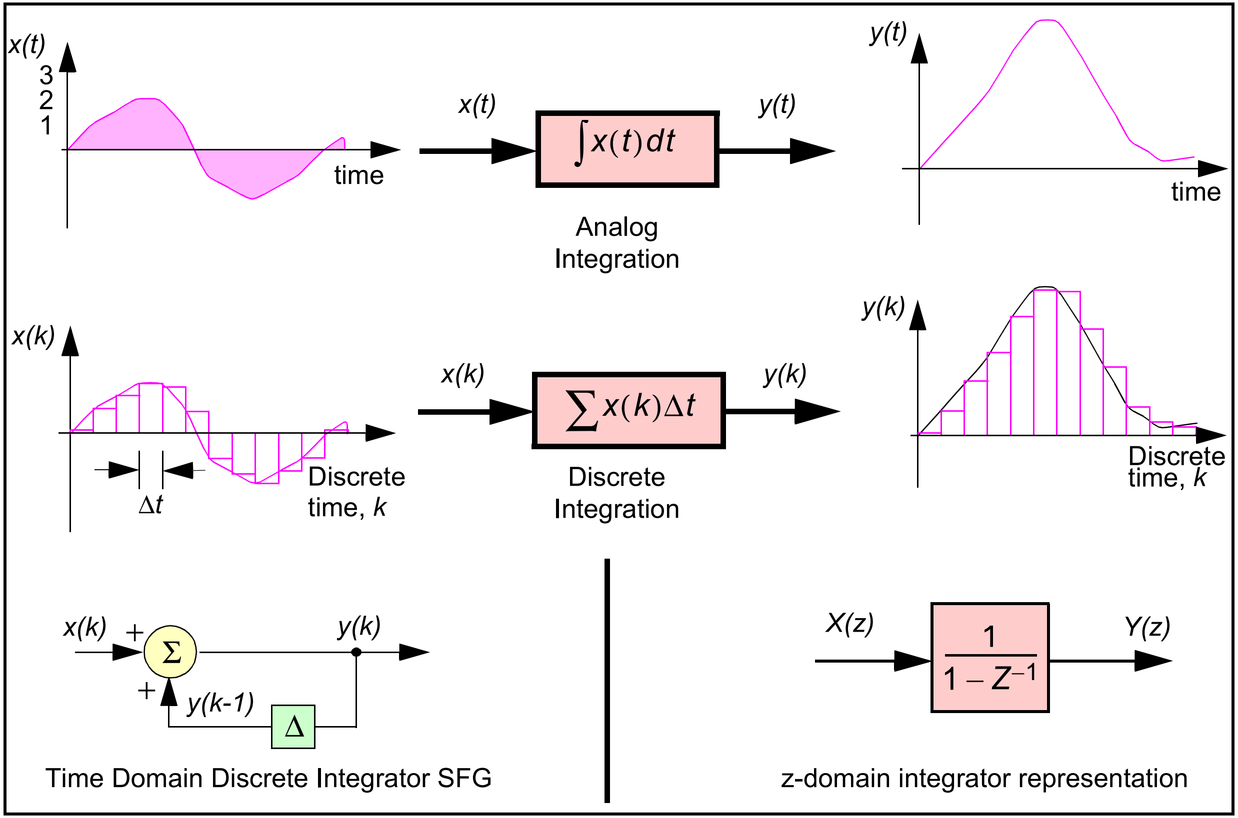
A digital *integrator* is essentially a one weight IIR filter:

$$y(k) = x(k) + y(k-1) \tag{111}$$

In the z-domain the transfer function of a discrete integrator is:

$$Y(z) = X(z) + z^{-1}Y(z)$$
$$\Rightarrow \quad \frac{Y(z)}{X(z)} = \frac{z}{z-1} \qquad (112)$$

When viewed in the frequency domain an integrator has the characteristics of a simple low pass filter.



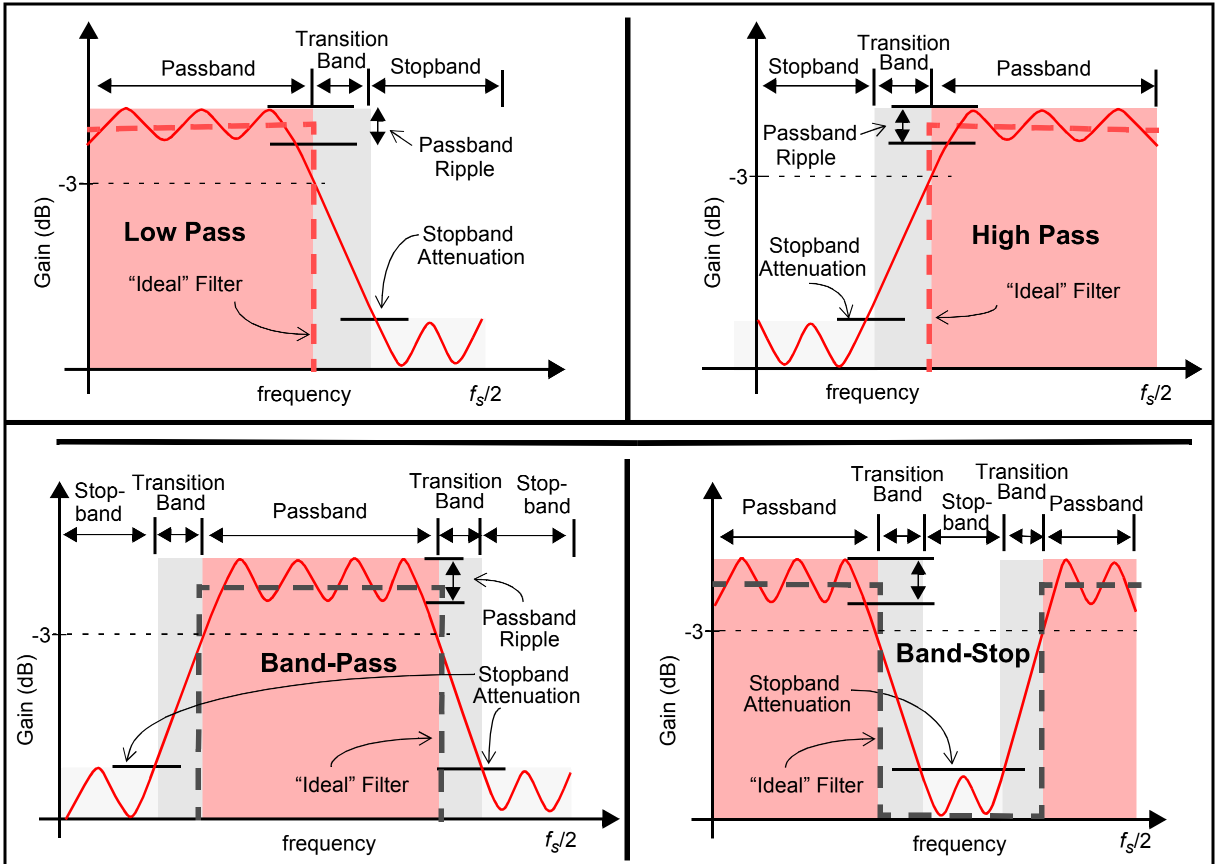
If a weight value of less than one is included in the feedback loop (say 0.99) then the integrator is often called a leaky integrator.

6 Filter Design Techniques

Design algorithms to find suitable weights for digital FIR filters are incorporated into SystemView and typically allow the user to specify the parameters of:

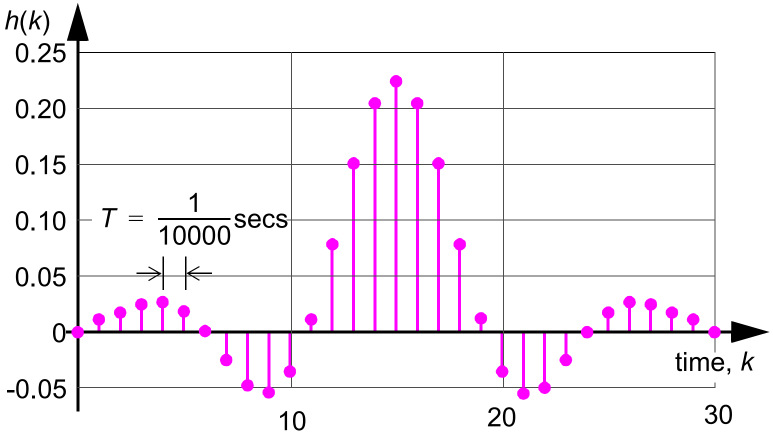
- Sampling frequency;
- Passband;
- Transition band;
- Stopband;
- Passband ripple;
- Stopband attenuation;
- No. of weights in the filter.

These parameters allow variations from the ideal (brick wall) filter, with the trade-offs being made by the design engineer. In general, the less stringent the bounds on the various parameters, then the fewer weights the *digital filter* will require:



Parameters for specifying low pass, high pass, band-pass and band stop filters

After the filter weights are produced by the SystemView design software the impulse response of the digital filter can be plotted, i.e. the filter weights shown against time:

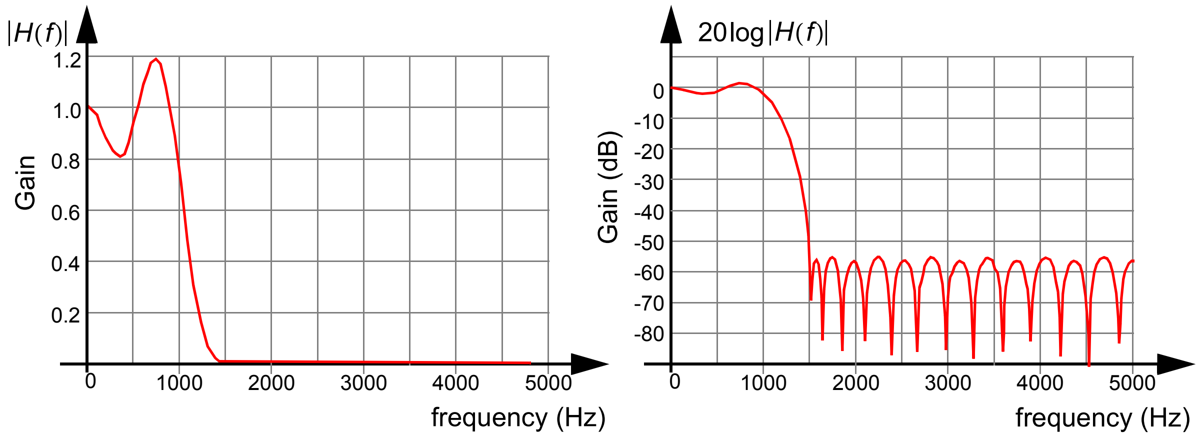


$w_0 = w_{30} = 0.00378...$
$w_1 = w_{29} = 0.00977...$
$w_2 = w_{28} = 0.01809...$
$w_3 = w_{27} = 0.02544...$
$w_4 = w_{26} = 0.027154...$
$w_5 = w_{25} = 0.019008...$
$w_6 = w_{24} = 0.00003...$
$w_7 = w_{23} = -0.02538...$
$w_8 = w_{22} = -0.04748...$
$w_9 = w_{21} = -0.05394...$
$w_{10} = w_{20} = -0.03487...$
$w_{11} = w_{19} = 0.01214...$
$w_{12} = w_{18} = 0.07926...$
$w_{13} = w_{17} = 0.14972...$
$w_{14} = w_{16} = 0.20316...$
$w_{15} = 0.22319...$
(Truncated to 5 decimal places)

DESIGN 1: Low Pass FIR Filter Impulse Response

The impulse response $h(k) = w_k$ of the low pass filter specified in the above SystemView dialog boxes: cut-off frequency 1000 Hz; passband gain 0 dB; stopband attenuation 60 dB; transition band 500 Hz; passband ripple 5 dB and sampling at f_s = 10000 Hz. The filter is linear phase and has 31 weights and therefore an impulse response of duration 31/10000 seconds. For this particular filter the weights are represented with floating point real numbers. Note that the filter was designed with 0 dB in the passband. As a quick check the sum of all of the coefficients is approximately 1, meaning that if a 0 Hz (DC) signal was input, the output is not amplified or attenuated, i.e. gain = 1 or 0 dB.

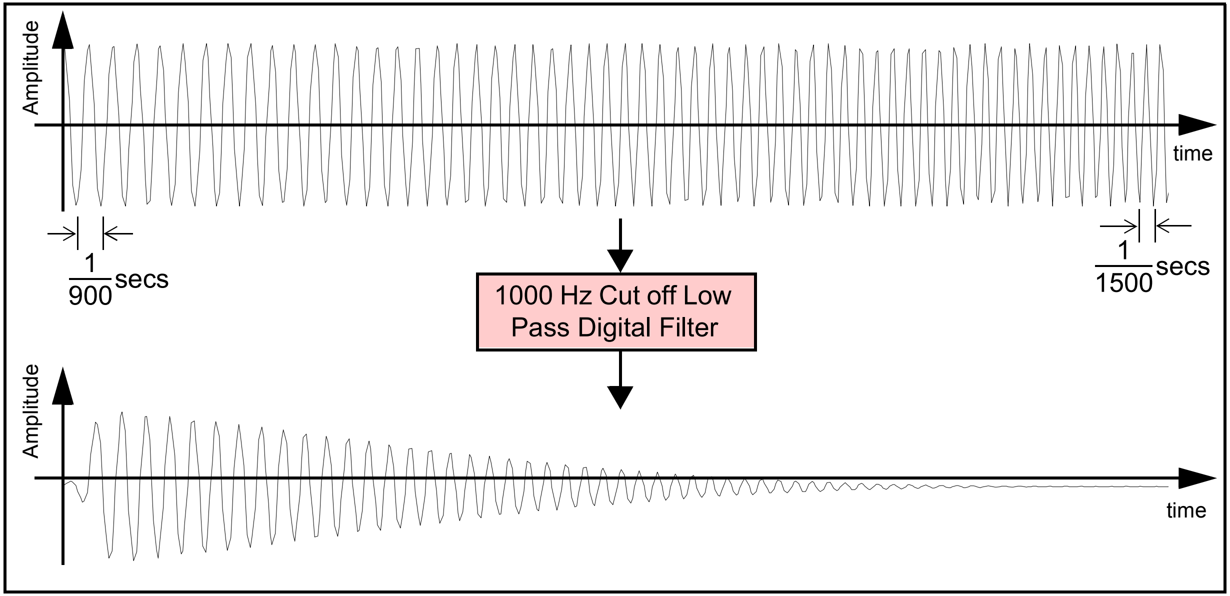
From the impulse response the DFT (or FFT) can be used to produce the filter magnitude frequency response and the actual filter characteristics can be compared with the original desired specification:



Linear Magnitude Response **Logarithmic Magnitude Response**

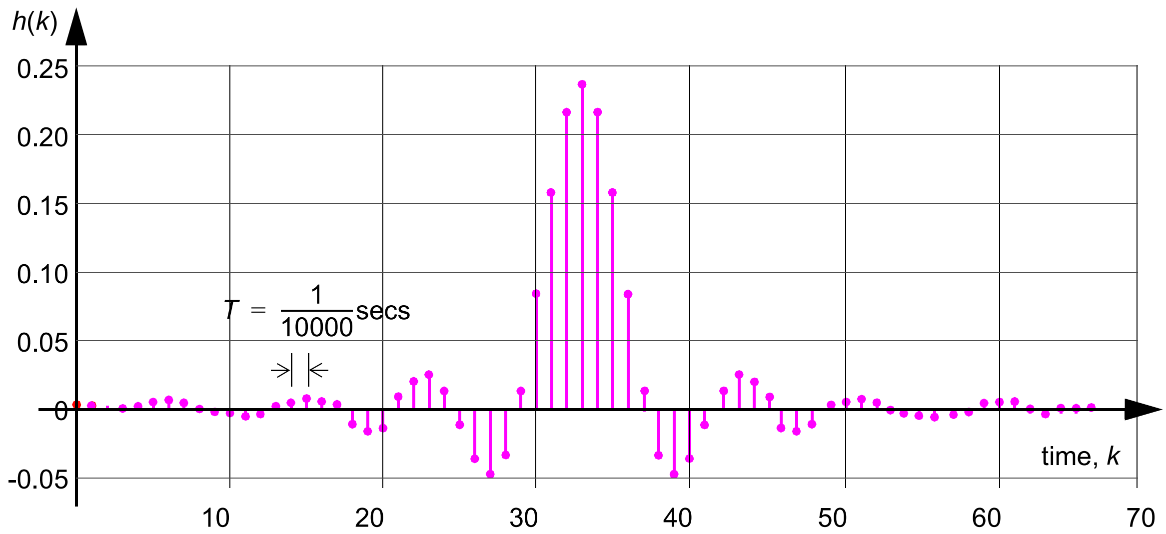
The 1024 point FFT (zero padded) of the above **DESIGN 1** low pass filter impulse response. The passband ripple is easier to see in the linear plot, whereas the stopband ripple is easier to see in the logarithmic plot.

To illustrate the operation of the above digital filter, a chirp signal starting at a frequency of 900 Hz, and linearly increasing to 1500 Hz over 0.05 seconds (500 samples) can be input to the filter and the output observed (individual samples are not shown):



As the chirp frequency reaches about 1000 Hz, the digital filter attenuates the amplitude output signal by a factor of around 0.7 (3dB) until at 1500 Hz the signal amplitude is attenuated by more than 60 dB or a factor of 0.001.

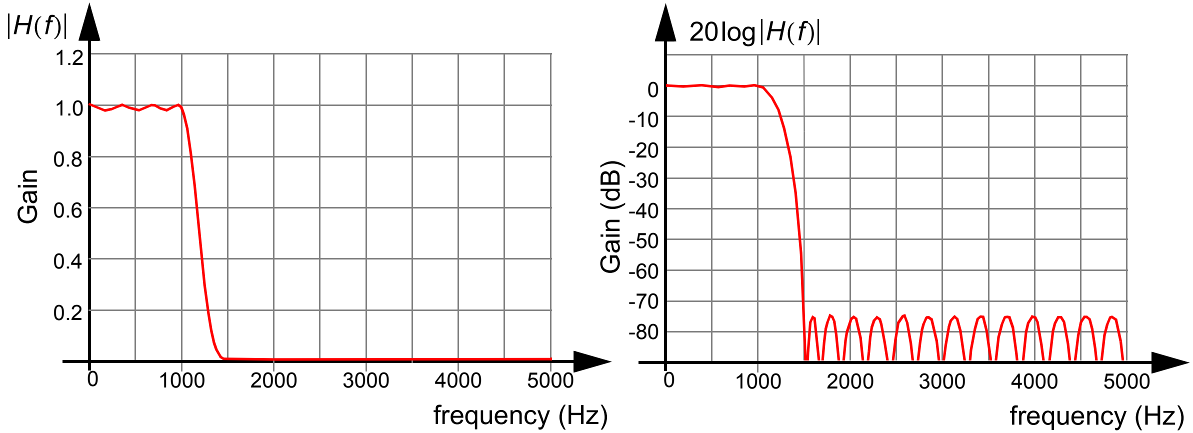
If a low pass filter with less passband ripple and a sharper cut off is required then another filter can be designed, although more weights will be required and the implementation cost of the filter has therefore increased. To illustrate this point, if the above low pass filter is redesigned, but this time with a stopband attenuation of 80dB, a passband ripple of 0.1dB and a transition band of, again, 500 Hz, the impulse response of the filter produced by the DSP design software now requires 67 weights:



DESIGN 2: Low Pass FIR Filter Impulse Response

The impulse response $h(k) = w_k$ of a low pass filter with: cut-off frequency 1000 Hz; passband gain 0 dB; stopband attenuation 80 dB; transition band 500 Hz; passband ripple 0.1 dB and sampling at f_s = 10000 Hz. The filter is linear phase and has 67 weights (compare to the above Design 1 which had 31 weights) and therefore an impulse response of duration 67/10000 seconds.

The frequency response of this Design 2 filter can be found by taking the FFT of the *digital filter* impulse response:



Linear Magnitude Response **Logarithmic Magnitude Response**

The 1024 point FFT (zero padded) of the **DESIGN 2** impulse response low pass filter impulse response. Note that, as specified, the filter roll-off is now steeper, the stopband is almost 80 dB and the inband ripple is only fractions of a dB.

Therefore low pass, high pass, bandpass, and bandstop *digital filters* can all be released by using the formal digital filter design methods that are available in a number of DSP software packages. (Or if you have a great deal of time on your hands you can design them yourself with a paper and pencil and reference to one of the classic DSP textbooks!) There are of course many filter design trade-offs. For example, as already illustrated above, to design a filter with a fast transition between stopband and passband requires more filter weights than a low pass filter with a slow roll-off in the transition band. However the more filter weights, the higher the computational load on the DSP processor, and the larger the group delay through the filter is likely to be. Care must therefore be taken to ensure that the computational load of the *digital filter* does not exceed the maximum processing rate of the DSP processor (which can be loosely measured in multiply-accumulates, MACs) being used to implement it. The minimum computation load of DSP processor implementing a digital filter in the time domain is at least:

$$\text{Computational Load of Digital Filter} = (\text{Sampling Rate} \times \text{No. of Filter Weights}) \text{ MACs} \tag{113}$$

and likely to be a factor greater than 1 higher due to the additional overhead of other assembly language instructions to read data in/out, to implement loops etc. Therefore a 100 weight digital filter sampling at 8000 Hz requires a computational load of 800,000 MACs/second (readily achievable in the mid-1990's), whereas for a two channel digital audio tape (DAT) system sampling at 48 kHz and using stereo *digital filters* with 1000 weights requires a DSP processor capable of performing almost 100 million MACs per second (verging on the "just about" achievable with late-1990s DSP processor technology).

6.1 Real Time Implementation

For each input sample, an FIR filter requires to perform N multiply accumulate (MAC) operations:

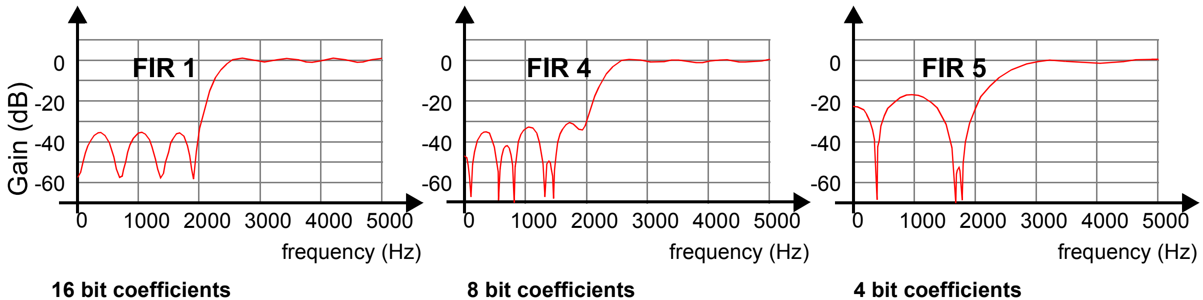
$$y(k) = \sum_{n=0}^{N-1} w_n x(k-n) \tag{114}$$

Therefore if a particular FIR filter is sampling data at f_s Hz, then the number of arithmetic operations per second is:

$$\text{MACs/sec} = Nf_s \tag{115}$$

6.2 FIR Filter Wordlength Considerations

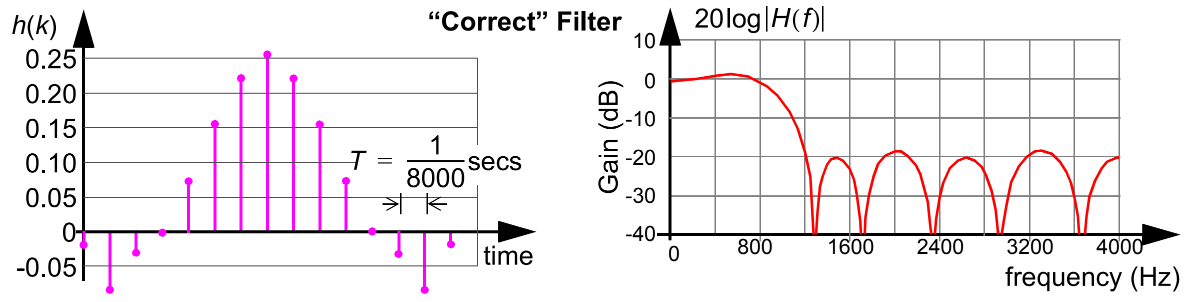
For a real time implementation of a digital filter, the wordlength used to represent the filter weights will of course have some bearing on the achievable accuracy of the frequency response. Consider for example the design of a high pass digital filter using 16 bit filter weights:



Low pass filters designed parameters: Transition Bandwidth = 500Hz; Passband Ripple = 1dB and stopband attenuations of 40 dB, 60 dB, and 80 dB.

6.3 Filtering Bit Errors

If we consider the possibility of a random single bit error in the weights of an FIR filter, the effect on the filter magnitude and phase response can be quite dramatic. Consider a simple 15 weight filter:

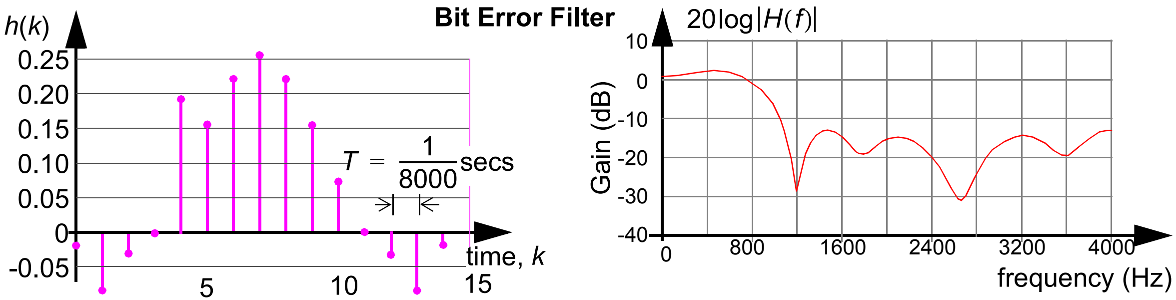


Fifteen weight low pass FIR filter cutting off at 800 Hz.

The 3rd coefficient is of value -0.0725..., and in 16 bit fractional binary notation this is $0.0001001010010 10_2$. If a single bit occurs in the 3rd bit of this binary coefficient then the value becomes:

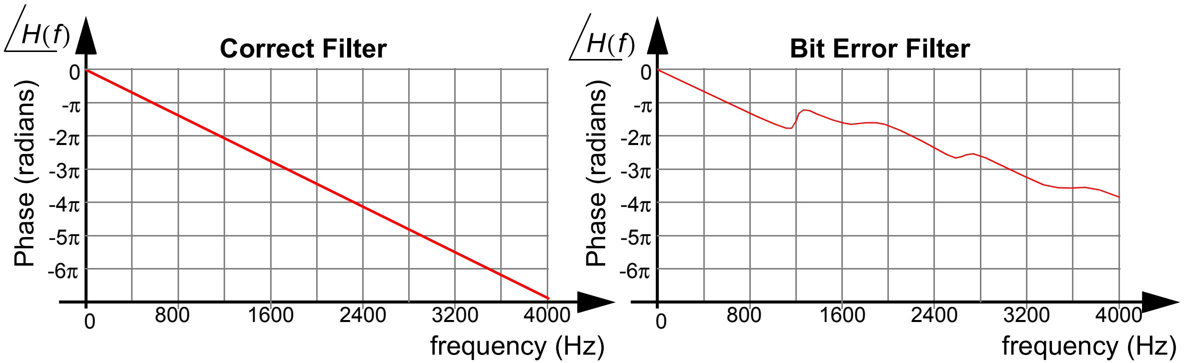
$$0.00\mathbf{1}1001010010 10_2 = -0.1957...$$

The impulse response clearly changes “a little” whereas the effect on the frequency response changes is a little more substantial and causes a loss of about 5 dB attenuation.



15 weights low pass FIR filter cutting off at 800 Hz with the 3rd coefficient being in error by a single bit. Note the change to the frequency response compared to the correct filter above.

Also because the impulse response is no longer symmetric the phase response is no longer linear:



Phase response of the original (“correct”) filter and the bit error filter. The result of the error in a single coefficient has caused the phase to be no longer exactly linear.

Of course the bit error may have occurred at the least significant bits and the frequency domain effect would be much less pronounced. However because of the excellent reliability of DSP processors the occurrence of bit errors in filter coefficients is unlikely.

7 Adaptive Filters

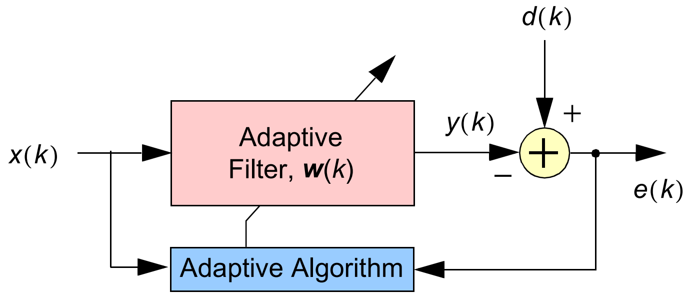
The discrete mathematics of adaptive filtering is based on the least squares minimization theory of the celebrated 19th Century German mathematician Gauss. Least squares is of course widely used in statistical analysis and virtually every branch of science and engineering. For many DSP applications, however, least squares minimization is applied to real time data, and therefore, presents the challenge of producing a real time implementation to operate on data arriving at high data rates (from 1k Hz to 100 kHz), and with *loosely* known statistics and properties. In addition, other cost functions besides least squares are also used.

One of the first suggestions of adaptive DSP algorithms was in Widrow and Hoff's classic paper on the adaptive switching circuits and the least mean squares (LMS) algorithm at the IRE WESCON Conference in 1960. This paper stimulated great interest by providing a practical and potentially real time solution for least squares implementation. Widrow followed up this work with two definitive and classic papers on *adaptive signal processing* in the 1970s.

Adaptive signal processing has found many applications. A generic breakdown of these applications can be made into the following categories of signal processing problems: signal detection (is it there?), signal estimation (what is it?), parameter or state estimation, signal compression, signal synthesis, signal classification, etc. The common attributes of *adaptive signal processing* applications include time varying (adaptive) computations (processing) using sensed input values (signals).

7.1 Generic Adaptive Signal Processor

The generic *adaptive filter* can be represented as:



$$y(k) = \text{Filter}\{x(k), \mathbf{w}(k)\}$$

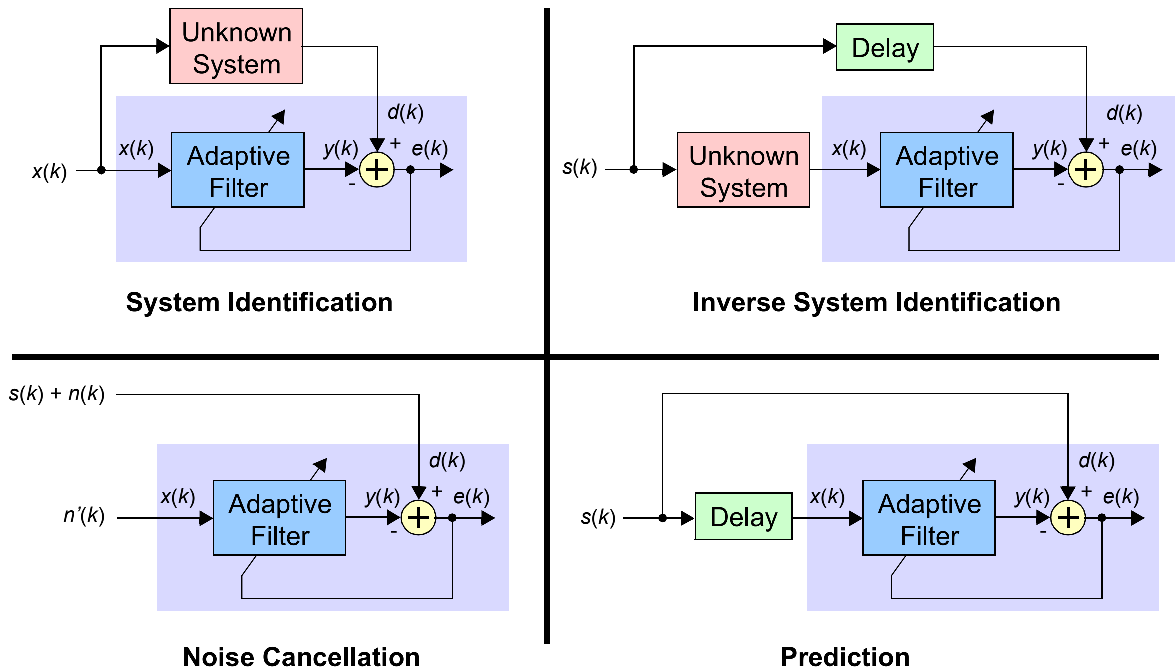
$$\mathbf{w}(k+1) = \mathbf{w}(k) + e(k)f\{d((k), x(k))\}$$

In the generic adaptive filter architecture the aim can intuitively be described as being to adapt the impulse response of the digital filter such that the input signal $x(k)$ is filtered to produce $y(k)$ which when subtracted from desired signal $d(k)$, will minimize the power of the error signal $e(k)$.

The adaptive filter output $y(k)$ is produced by the filter weight vector, $\mathbf{w}(k)$, convolved (in the linear case) with $x(k)$. The adaptive filter weight vector is updated based on a function of the error signal $e(k)$ at each time step k to produce a new weight vector, $\mathbf{w}(k+1)$ to be used at the next time step. This adaptive algorithm is used in order that the input signal of the filter, $x(k)$, is filtered to produce an output, $y(k)$, which is *similar* to the desired signal, $d(k)$, such that the power of the error signal, $e(k) = d(k) - y(k)$, is minimized. This minimization is essentially achieved by exploiting the correlation that should exist between $d(k)$ and $y(k)$.

The adaptive digital filter can be an FIR, IIR, Lattice or even a non-linear (Volterra) filter, depending on the application. The most common by far is the FIR. The adaptive algorithm can be based on gradient techniques such as the LMS, or on recursive least squares techniques such as the RLS. In general different algorithms have different attributes in terms of minimum error achievable, convergence time, and stability.

There are at least four general architectures that can be set up for *adaptive filters*: (1) System identification; (2) Inverse system identification; (3) Noise cancellation; (4) Prediction. Note that all of these architectures have the same generic adaptive filter as shown below (the “Adaptive Algorithm” block explicitly drawn above has been left out for illustrative convenience and clarity):



Four *adaptive signal processing* architectures

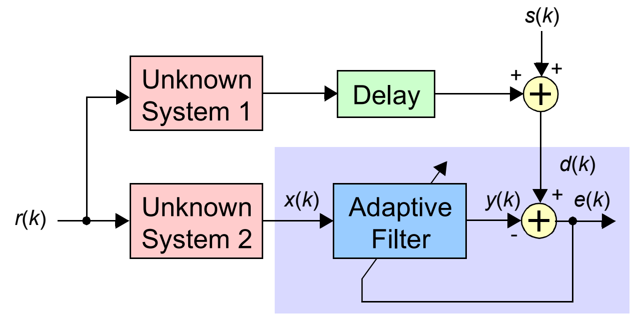
Consider first the system identification; at an intuitive level, if the adaptive algorithm is indeed successful at minimizing the error to zero, then by simple inspection, the transfer function of the “Unknown System” must be identical to the transfer function of the adaptive filter. Given that the error of the adaptive filter is now zero, then the adaptive filters weights are no longer updated and will remain in a steady state. As long as the unknown system does not change its characteristics we have now successfully identified (or modelled) the system. If the adaption was not perfect and the error is "very small" rather than zero (which is more likely in real applications) then it is fair to say the we have a good model rather than a perfect model.

Similarly, for the inverse system identification, if the error adapts to zero over a period of time, then by observation, the transfer function of the adaptive filter must be the exact inverse of the "Unknown System". (Note that the "Delay" is necessary to ensure that the problem is causal and therefore solvable with real systems, i.e. given that the "Unknown System" may introduce a time delay in producing $x(k)$, then if the "Delay" was not present in the path of the desired signal, the system would be required to produce an anti-delay or look ahead in time - clearly this is impossible.)

For the noise cancellation architecture, if the input signal is $s(k)$ which is corrupted by additive noise, $n(k)$, then the aim is to use a correlated noise reference signal, $n'(k)$ as an input to the adaptive filter, such that when performing the adaption there is only information available to implicitly model the noise signal, $n(k)$ and therefore when this filter adapts to a steady state we would expect that $e(k) \approx s(k)$.

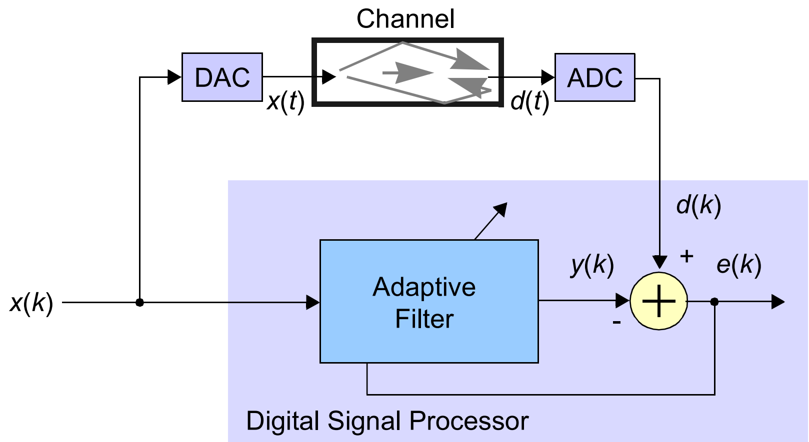
Finally, for the prediction filter, if the error is set to be adapted to zero, then the adaptive filter must predict future elements of the input $s(k)$ based only on past observations. This can be performed if the signal $s(k)$ is periodic and the filter is long enough to "remember" past values. One application therefore of the prediction architecture could be to extract periodic signals from stochastic noise signals. The prediction filter can be extended to a "smoothing filter" if data are processed off-line -- this means that samples before and after the present sample are filtered to obtain an estimate of the present sample. Smoothing cannot be done in real-time, however there are important applications where real-time processing is not required (e.g., geophysical seismic signal processing).

A particular application may have elements of more than one single architecture, for example in the following, if the adaptive filter is successful in modelling "Unknown System 1", and inverse modelling "Unknown System 2", then if $s(k)$ is uncorrelated with $r(k)$ then the error signal is likely to be $e(k) \approx s(k)$:



An *adaptive filtering* architecture incorporating elements of system identification, inverse system identification and noise cancellation

In the four general architectures shown above the unknown systems being investigated will normally be analog in nature, and therefore suitable ADCs and DACs would be used at the various analog input and output points as appropriate. For example, if an adaptive filter was being used to find a model of a baseband communications channel the overall hardware set up would be:



The analog-digital interfacing for a system identification, or modelling, of a baseband communications channel.

7.2 Least Squares

Adaptive signal processing is fundamentally based on least squares techniques which can be cast as the problem of solving an overdetermined linear set of equations, $\mathbf{A}\mathbf{x} = \mathbf{b}$. $\mathbf{A}$ is a known $m \times n$ matrix of rank n (with $m > n$), $\mathbf{b}$ is a known m element vector, and $\mathbf{x}$ is an unknown n element vector, then the *least squares* solution is given by:

$$\boldsymbol{x}_{LS} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b} \tag{116}$$

(Note that if the problem is underdetermined, $m < n$, then Eq. 116 is not the solution, and in fact there is no unique solution; a good (i.e., close) solution can often be found however using the pseudoinverse obtained via singular value decomposition.)

The *least squares* solution can be derived as follows. Consider again the overdetermined linear set of equations:

$$\underbrace{\begin{bmatrix} a_1 & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ a_{31} & a_{32} & \dots & a_{3n} \\ a_{41} & a_{42} & \dots & a_{4n} \\ : & : & \dots & : \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ : \\ x_n \end{bmatrix}}_{\boldsymbol{x}} = \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ : \\ b_m \end{bmatrix}}_{\boldsymbol{b}} \tag{117}$$

If $\boldsymbol{A}$ is a nonsingular square matrix, i.e. $m = n$, then the solution can be calculated as:

$$\boldsymbol{x} = \boldsymbol{A}^{-1}\boldsymbol{b} \tag{118}$$

However if $m \neq n$ then $\boldsymbol{A}$ is a rectangular matrix and therefore not invertible, and the above equation cannot be solved to give exact solution for $\boldsymbol{x}$. If $m < n$ then the system is often referred to as underdetermined and an infinite number of solutions exist for $\boldsymbol{x}$ (as long as the m equations are consistent). If $m > n$ then the system of equations is overdetermined and we can look for a solution by striving to make $\boldsymbol{Ax}$ be as close as possible to $\boldsymbol{b}$, by minimizing $\boldsymbol{Ax} - \boldsymbol{b}$ in some sense. The most mathematical tractable way to do this is by the method of *least squares*, performed by minimizing the 2-norm denoted by e:

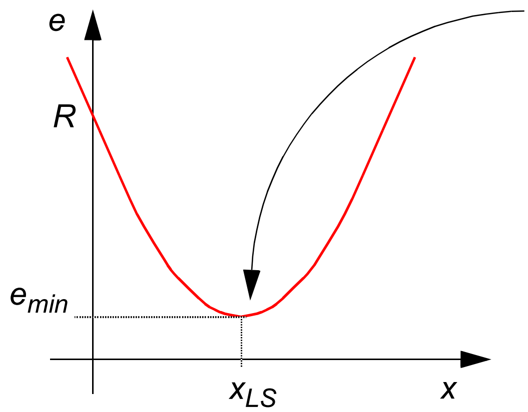
$$e = (\|\boldsymbol{Ax} - \boldsymbol{b}\|_2)^2 = (\boldsymbol{Ax} - \boldsymbol{b})^T(\boldsymbol{Ax} - \boldsymbol{b}) \tag{119}$$

Plotting e against the n-dimensional vector $\boldsymbol{x}$ gives a hyperparabolic surface in $(n+1)$-dimensions. If $n = 1$, $\boldsymbol{x}$ has only one element and the surface is a simple parabola. For example consider the case where $\boldsymbol{A}$ is a 2×1 matrix, then from Eq. 119:

$$\begin{aligned} e &= \left(\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} x - \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}\right)^T \left(\begin{bmatrix} a_1 \\ a_2 \end{bmatrix} x - \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}\right) \\ &= \begin{bmatrix} a_1 & a_2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} x^2 - 2 \begin{bmatrix} b_1 & b_2 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} x + \begin{bmatrix} b_1 & b_2 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} \\ &= \left[a_1^2 + a_2^2\right] x^2 - 2\left[a_1 b_1 + a_2 b_2\right] x + \left[b_1^2 + b_2^2\right] \\ &= Px^2 - Qx + R \end{aligned} \tag{120}$$

where $P = \left[a_1^2 + a_2^2\right]$, $Q = 2\left[a_1 b_1 + a_2 b_2\right]$ and $R = \left[b_1^2 + b_2^2\right]$.

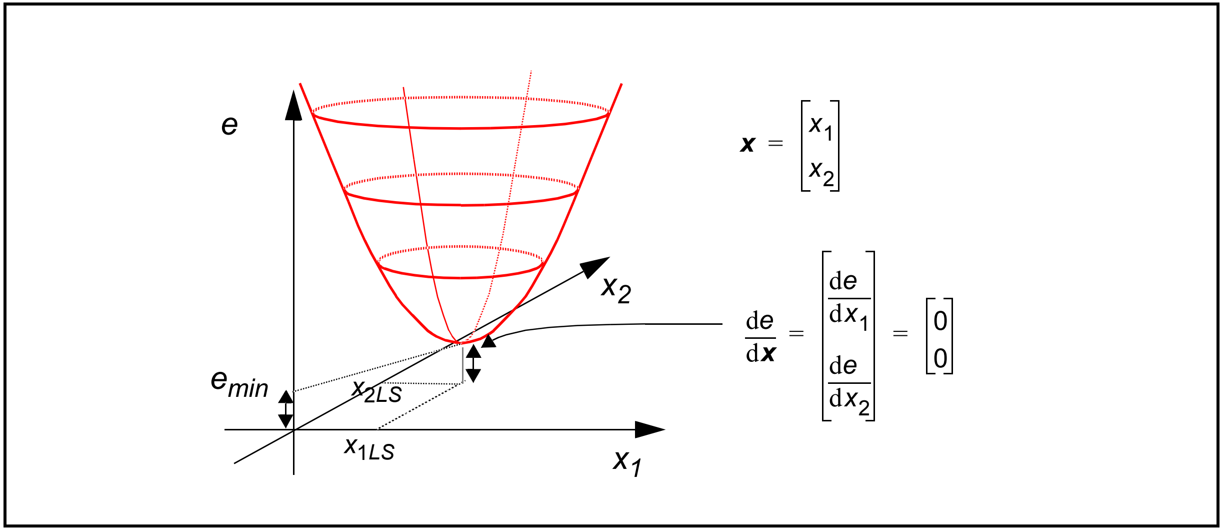
Clearly the minimum point on the surface lies at the bottom of the parabola:



$$\frac{de}{dx} = 2Px_{LS} - Q = 0 \quad \Rightarrow \quad x_{LS} = \frac{Q}{2P}$$

$$x_{LS} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{b} = \frac{[a_1b_1 + a_2b_2]}{[a_1^2 + a_2^2]} = \frac{Q}{2P}$$

If $n = 2$, $\boldsymbol{x} = [x_1 \; x_2]^T$ and the error surface is a paraboloid. This surface has one minimum point at the bottom of the paraboloid where the gradient of the surface with respect to both x_1 and x_2 axis is zero:



If the $\boldsymbol{x}$ vector has three or more elements ($n \geq 3$) the surface will be in four or more dimensions and cannot be shown diagrammatically.

To find the minimum value of e for the general case of an n-element $\boldsymbol{x}$ vector the "bottom" of the hyperparaboloid can be found by finding the point where the gradient in every dimension is zero (cf. the above 1 and 2-dimensioned examples). Therefore differentiating e with respect to the vector $\boldsymbol{x}$:

$$\frac{\mathrm{d}e}{\mathrm{d}\boldsymbol{x}} = \left[\frac{\mathrm{d}e}{\mathrm{d}x_1}\ \frac{\mathrm{d}e}{\mathrm{d}x_2}\ \frac{\mathrm{d}e}{\mathrm{d}x_3}\ \ \dots\ \ \frac{\mathrm{d}e}{\mathrm{d}x_n}\right]^T = 2\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{x}-\boldsymbol{b}) \tag{121}$$

and setting the gradient vector to the zero vector,

$$\frac{\mathrm{d}e}{\mathrm{d}\boldsymbol{x}} = \mathbf{0} \tag{122}$$

to find the minimum point, e_{min}, on the surface gives the least squares error solution for $\boldsymbol{x}_{LS}$:

$$\begin{aligned} 2\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{x}_{LS}-\boldsymbol{b}) &= \mathbf{0} \\ \boldsymbol{A}^T\boldsymbol{A}\boldsymbol{x}_{LS}-\boldsymbol{A}^T\boldsymbol{b} &= \mathbf{0} \\ \boldsymbol{x}_{LS} &= (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b} \end{aligned} \tag{123}$$

If the rank of matrix $\boldsymbol{A}$ is less than n, then the inverse matrix $(\boldsymbol{A}^T\boldsymbol{A})^{-1}$ does not exist and the least squares solution cannot be found using Eq. 123 and the pseudoinverse requires to be calculated using singular value decomposition techniques. Note that if $\boldsymbol{A}$ is an invertible square matrix, then the *least squares* solution simplifies to:

$$\boldsymbol{x} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b} = \boldsymbol{A}^{-1}\boldsymbol{A}^{-T}\boldsymbol{A}^T\boldsymbol{b} = \boldsymbol{A}^{-1}\boldsymbol{b} \tag{124}$$

7.3 Least Squares Residual

The least squares error solution to the overdetermined system of equations, $\boldsymbol{Ax} = \boldsymbol{b}$, is given by:

$$\boldsymbol{x}_{LS} = (\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T\boldsymbol{b} \tag{125}$$

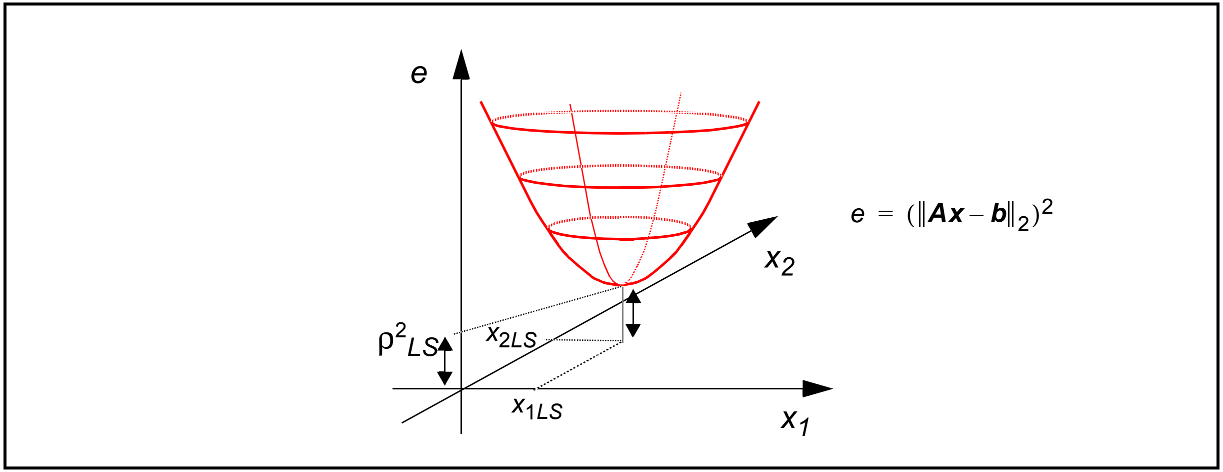
where $\boldsymbol{A}$ is a known $m \times n$ matrix of rank n and with $m > n$, $\boldsymbol{b}$ is a known m element vector, and $\boldsymbol{x}$ is an unknown n element vector. The *least squares residual* given by:

$$\boldsymbol{r}_{LS} = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_{LS} \tag{126}$$

is a measure of the error obtained when using the method of least squares. The smaller the value of $\boldsymbol{r}_{LS}$, then the more accurately $\boldsymbol{b}$ can be generated from the columns of the matrix $\boldsymbol{A}$. The magnitude or size of the *least squares residual* is calculated by finding the squared magnitude, or 2-norm of $\boldsymbol{r}_{LS}$:

$$\rho_{LS} = \|\boldsymbol{A}\boldsymbol{x}_{LS} - \boldsymbol{b}\|_2 \tag{127}$$

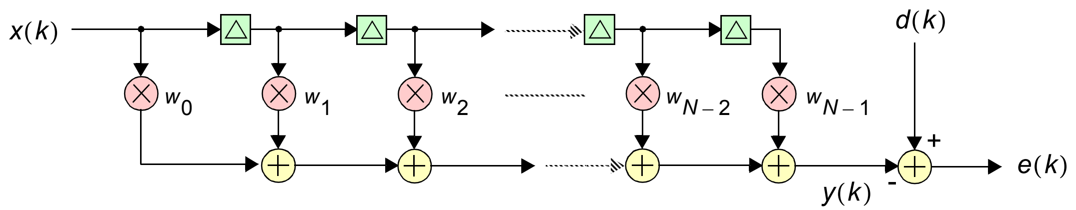
As an example, for a system with $n = 2$ the *least squares residual* can be shown on the least squares error surface, e, as:



Note that if $m = n$, and $\boldsymbol{A}$ is a non-singular matrix, then $\rho_{LS} = 0$.

7.4 Wiener-Hopf Solution

Consider the following architecture based on a FIR filter and a subtraction element:



The output of an FIR filter, $y(k)$ is subtracted from a desired signal, $d(k)$ to produce an error signal, $e(k)$. If there is some correlation between the input signal, $x(k)$ and the desired signal, $d(k)$ then values can be calculated for the filter weights, $w(0)$ to $w(N-1)$ in order to minimize the mean squared error, $E\{e^2(k)\}$.

If the signal $x(k)$ and $d(k)$ are in some way correlated, then certain applications and systems may require that the digital filter weights, $w(0)$ to $w(N-1)$ are set to values such that the power of the error signal, $e(k)$ is minimised. If weights are found that minimize the error power in the mean squared sense, then this is often referred to as the Wiener-Hopf solution.

To derive the Wiener Hopf solution it is useful to use a vector notation for the input vector and the weight vector. The output of the filter, $y(k)$, is the convolution of the weight vector and the input vector:

$$y(k) = \sum_{n=0}^{N-1} w_n x(k-n) = \mathbf{w}^T \mathbf{x}(k) \qquad (128)$$

where,

$$\mathbf{w} = [w_0 \ \ w_1 \ \ w_2 \ \ \ldots \ w_{N-2} \ \ w_{N-1}]^T \tag{129}$$

and,

$$\mathbf{x}(k) = [x(k) \ \ x(k-1) \ \ x(k-2) \ \ \ldots \ x(k-N+2) \ \ x(k-N+1)]^T \tag{130}$$

Assuming that $x(k)$ and $d(k)$ are wide sense stationary processes and are correlated in some sense, then the error, $e(k) = d(k) - y(k)$ can be minimised in the mean squared sense.

To derive the Wiener-Hopf equations consider first the squared error:

$$\begin{aligned} e^2(k) &= [d(k) - y(k)]^2 \\ &= d^2(k) - [\mathbf{w}^T\mathbf{x}(k)]^2 - 2d(k)\mathbf{w}^T\mathbf{x}(k) \\ &= d^2(k) - \mathbf{w}^T\mathbf{x}(k)\mathbf{x}^T(k)\mathbf{w} - 2\mathbf{w}^T d(k)\mathbf{x}(k) \end{aligned} \tag{131}$$

Taking expected (or mean) values we can write the mean squared error (MSE), $E\{e^2(k)\}$ as:

$$E\{e^2(k)\} = E\{d^2(k)\} - \mathbf{w}^T E\{\mathbf{x}(k)\mathbf{x}^T(k)\}\mathbf{w} - 2\mathbf{w}^T E\{d(k)\mathbf{x}(k)\} \tag{132}$$

Writing in terms of the $N \times N$ correlation matrix,

$$\mathbf{R} = E\{\mathbf{x}(k)\mathbf{x}^T(k)\} = \begin{bmatrix} r_0 & r_1 & r_2 & \dots & r_{N-1} \\ r_1 & r_0 & r_1 & \dots & r_{N-2} \\ r_2 & r_1 & r_0 & \dots & r_{N-3} \\ : & : & : & \dots & : \\ r_{N-1} & r_{N-2} & r_{N-3} & \dots & r_o \end{bmatrix} \tag{133}$$

and the $N \times 1$ cross correlation vector,

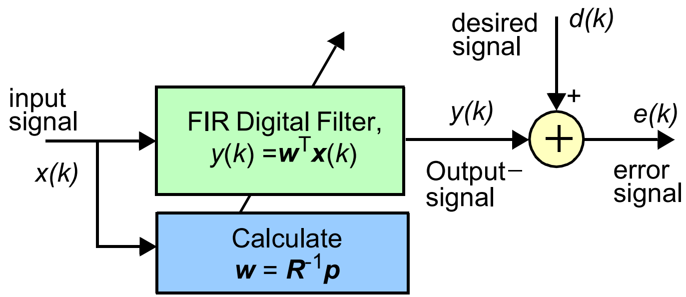
$$\mathbf{p} = E\{d(k)\mathbf{x}(k)\} = \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ : \\ p_{N-1} \end{bmatrix} \tag{134}$$

gives,

$$\zeta = E\{e^2(k)\} = E\{d^2(k)\} + \mathbf{w}^T\mathbf{R}\mathbf{w} - 2\mathbf{w}^T\mathbf{p} \tag{135}$$

where ζ is used for notational convenience to denote the MSE performance surface. Given that this equation is quadratic in $\mathbf{w}$ then there is only one minimum value. The minimum mean squared error (MMSE) solution, $\mathbf{w}_{opt}$, can be found by setting the (partial derivative) gradient vector, ∇, to zero:

$$\nabla = \frac{\partial \zeta}{\partial \mathbf{w}} = 2\mathbf{R}\mathbf{w} - 2\mathbf{p} = \mathbf{0}$$
$$\Rightarrow \mathbf{w}_{opt} = \mathbf{R}^{-1}\mathbf{p} \tag{136}$$

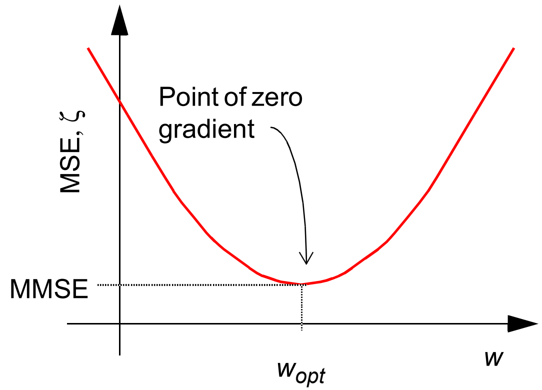


A simple block diagram for the Wiener-Hopf calculation. Note that there is no feedback and therefore, assuming $\mathbf{R}$ is non-singular, the algorithm is unconditionally stable.

To appreciate the quadratic and single minimum nature of the error performance surface consider the trivial case of a one weight filter:

$$\zeta = E\{d^2(k)\} + rw^2 - 2wp \tag{137}$$

where $E[d^2(k)]$, r, and p are all constant scalars. Plotting mean squared error (MSE), ζ, against the weight vector, w, produces a parabola (upfacing):



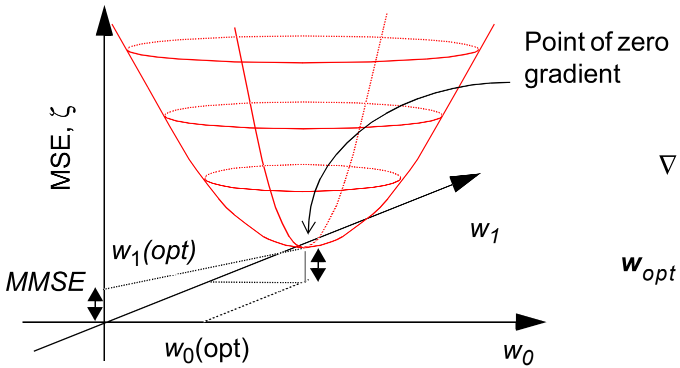
$$\nabla = \frac{d\zeta}{dw} = 2rw - 2p = 0$$

$$w_{opt} = r^{-1}p$$

The mean square error (MSE) performance surface, ζ, of for a single weight filter.

The MMSE solution occurs when the surface has gradient, $\nabla = 0$.

If the filter has two weights the performance surface is a paraboloid which can be drawn in 3 dimensions:



$$\nabla = \frac{d\zeta}{d\boldsymbol{w}} = 2\boldsymbol{R}\boldsymbol{w} - 2\boldsymbol{p} = 0$$

$$\boldsymbol{w}_{opt} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}_{opt} = \begin{bmatrix} r_0 & r_1 \\ r_1 & r_0 \end{bmatrix}^{-1} \begin{bmatrix} p_0 \\ p_1 \end{bmatrix}$$

The mean square error (MSE) performance surface, ζ, of for a two weight filter.

If the filter has more than three weights then we cannot draw the performance surface in three dimensions, however, mathematically there is only one minimum point which occurs when the gradient vector is zero. A performance surface with more than three dimensions is often called a hyperparaboloid.

To actually calculated the Wiener-Hopf solution, $\boldsymbol{w}_{opt} = \boldsymbol{R}^{-1}\boldsymbol{p}$ requires that the $\boldsymbol{R}$ matrix and $\boldsymbol{p}$ vector are realised from the data $x(k)$ and $d(k)$, and the $\boldsymbol{R}$ matrix is then inverted prior to premultiplying vector $\boldsymbol{p}$. Given that we assumed that $x(k)$ and $d(k)$ are stationary and ergodic, then we can estimate all elements of $\boldsymbol{R}$ and $\boldsymbol{p}$ from:

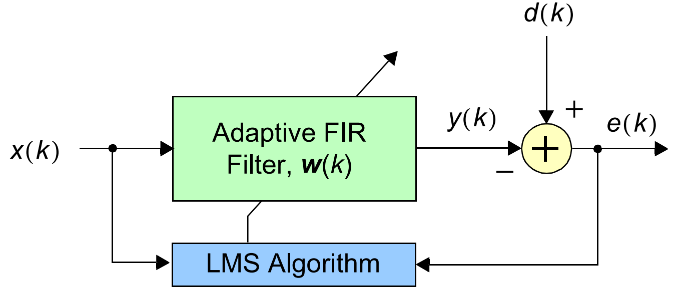
$$r_n = \frac{1}{M}\sum_{i=0}^{M-1} x_i x_{i+n} \qquad \text{and} \qquad p_n = \frac{1}{M}\sum_{i=0}^{M-1} d_i x_{i+n} \tag{138}$$

Calculation of $\boldsymbol{R}$ and $\boldsymbol{p}$ requires approximately $2MN$ multiply and accumulate (MAC) operations where M is the number of samples in a "suitably" representative data sequence, and N is the adaptive filter length. The inversion of $\boldsymbol{R}$ requires around N^3 MACs, and the matrix-vector multiplication, N^2 MACs. Therefore the total number of computations in performing this one step algorithm is $2MN + N^3 + N^2$ MACs. The computation load is therefore very high and real time operation is computationally expensive. More importantly, if the statistics of signals $x(k)$ or $d(k)$ change, then the filter weights will need to be recalculated, i.e. the algorithm has no tracking capabilities. Hence direct implementation of the Wiener-Hopf solution is not practical for real time DSP implementation because of the high computational load, and the need to recalculate when the signal statistics change. For this reason real time systems which need to minimize an error signal power use gradient descent based adaptive filters such as the least mean squares (LMS) or recursive least squares (RLS) type algorithms.

7.5 Least Mean Squares (LMS) Algorithm

The LMS is an adaptive signal processing algorithm that is very widely used in adaptive signal processing applications such as system identification, inverse system identification, noise cancellation and prediction. The *LMS algorithm* is very simple to implement in real time and in the mean will adapt to a neighbourhood of the Wiener-Hopf least mean square solution. The *LMS algorithm* can be summarised as follows:

To derive the *LMS algorithm*, first consider plotting the mean squared error (MSE) performance surface (i.e. $E\{e^2(k)\}$ as a function of the weight values) which gives an $(N+1)$-dimensional hyperparaboloid which has one minimum. It is assumed that $x(k)$ (the input data sequence) and



- $y(k) = \sum_{n=0}^{N-1} w(k)x(k-n) = \mathbf{w}^T(k)\mathbf{x}(k)$

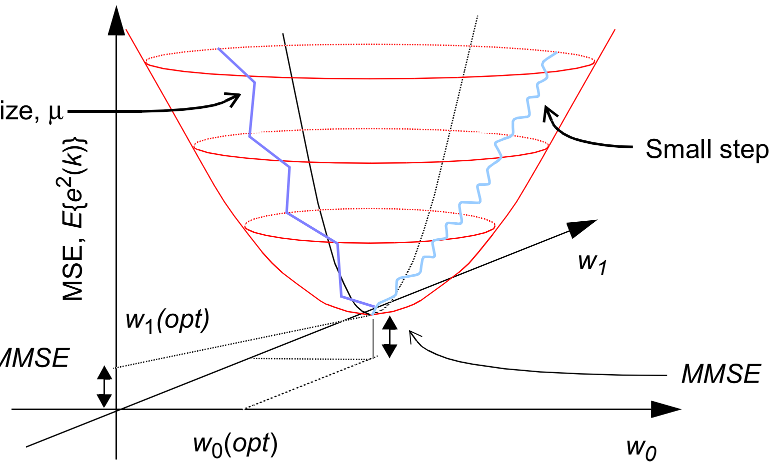
  where
  $$\mathbf{x}(k) = [x(k), x(k-1), x(k-2), \ldots, x(k-N+2), x(k-N+1)]^T$$
  $$\mathbf{w}(k) = [w_0(k), w_1(k), w_2(k), \ldots, w_{N-2}(k), w_{N-1}(k)]^T$$

- $e(k) = d(k) - y(k) = d(k) - \mathbf{w}^T(k)\mathbf{x}(k)$

- $\mathbf{w}(k+1) = \mathbf{w}(k) + 2\mu e(k)\mathbf{x}(k)$

In the generic adaptive filtering architecture the aim can intuitively be described as adapting the impulse response of the FIR digital filter such that the input signal $x(k)$ is filtered to produce $y(k)$ which, when subtracted from desired signal $d(k)$, will minimizethe error signal $e(k)$. If the filter weights are updated using the LMS weight update then the adaptive FIR filter will adapt to the minimum mean squared error, assuming $d(k)$ and $x(k)$ to be wide sense stationary signals.

$d(k)$ (a desired signal) are wide sense stationary signals. For discussion and illustration purposes the three dimensional paraboloid for a two weight FIR filter can be drawn:



The mean square error (MSE) performance surface for a two weight FIR filter. The Wiener-Hopf solution is denoted as $w_k(opt))$, which denotes where the minimum MSE (MMSE) occurs. The gradient based LMS algorithm will (on average) adapt towards the MMSE by taking “jumps” in the direction of the negative of the gradient of the surface (therefore “downhill”).

To find the minimum mean squared error (MMSE) we can use the Wiener Hopf equation, however this is an expensive solution in computation terms. As an alternative we can use gradient based techniques, whereby we can traverse down the inside of the parabola by using an iterative algorithm which always updates the filter weights in the direction opposite of the steepest gradient. The iterative algorithm is often termed gradient descent and has the form:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \mu(-\underline{\nabla}_k) \tag{139}$$

where $\underline{\nabla}_k$ is the gradient of the performance surface:

$$\begin{aligned}\underline{\nabla}_k &= \frac{\partial}{\partial \boldsymbol{w}(k)} E\{e^2(k)\} \\ &= 2\boldsymbol{R}\boldsymbol{w}(k) - 2\boldsymbol{p}\end{aligned} \qquad (140)$$

where $\boldsymbol{R}$ is the correlation matrix, $\boldsymbol{p}$ is cross correlation vector and μ is the step size (used to control the speed of adaption and the achievable minimum or misadjustment). In the above figure a small step size "jumps" in small steps towards the minimum and is therefore slow to adapt, however the small jumps mean that it will arrive very close to the MMSE and continue to jump back and forth close to the minimum. For a large step size the jumps are larger and adaption to the MMSE is faster, however when the weight vector reaches the bottom of the bowl it will jump back and forth around the MMSE with a large magnitude than for the small step size. The error caused by the traversing of the bottom of the bowl is usually called the excess mean squared error (EMSE).

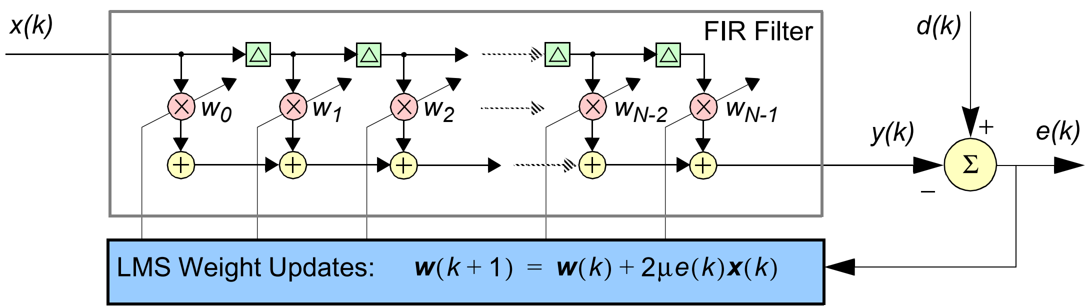
To calculate the MSE performance surface gradient directly is (like the Wiener Hopf equation) very expensive as it requires that both $\boldsymbol{R}$, the correlation matrix and $\boldsymbol{p}$, the cross correlation vector are known. In addition, if we knew $\boldsymbol{R}$ and $\boldsymbol{p}$, we could directly compute the optimum weight vector. But in general, we do not have access to $\boldsymbol{R}$ and $\boldsymbol{p}$. Therefore a subtle innovation, first defined for DSP by Widrow et al., was to replace the actual gradient with an instantaneous (noisy) gradient estimate. One approach to generating this noisy gradient estimate is to take the gradient of the actual squared error (versus the mean squared error), i.e.

$$\begin{aligned}\hat{\underline{\nabla}}_k &= \frac{\partial}{\partial \mathbf{w}(k)} e^2(k) \\ &= 2e(k)\frac{\partial}{\partial \mathbf{w}(k)} e(k) = -2e(k)\frac{\partial}{\partial \mathbf{w}(k)} y(k) = -2e(k)\mathbf{x}(k)\end{aligned} \tag{141}$$

Therefore using this estimated gradient, $\hat{\underline{\nabla}}_k$ in the gradient descent equation yields the *LMS algorithm*:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + 2\mu e(k)\mathbf{x}(k) \tag{142}$$

The LMS is very straightforward to implement and only requires N multiply-accumulates (MACs) to perform the FIR filtering, and N MACs to implement the LMS equation. A typical signal flow graph for the LMS is shown below:



A simple signal flow graph for an adaptive FIR filter, where the adaptive nature of the filter weights is explicitly illustrated.

From a practical implementation point of view the algorithm designer must carefully choose the filter length to suit the application. In addition, the step size must be chosen to ensure stability and a good convergence rate. For the LMS upper and lower bounds for the adaptive step size can be calculated as:

$$0 < \mu < \frac{1}{NE\{x^2(k)\}} \cong \qquad 0 < \mu < \frac{1}{N\{\text{Input Signal Power}\}} \tag{143}$$

A more formal bound can be defined in terms of the eigenvalues of the input signal correlation matrix. However for practical purposes these values are not calculated and the above practical bound is used.

In general the speed of adaption is inversely proportional to the step size, and the excess MSE or steady state error is proportional to the step size. Clearly a trade-off exists -- once again the responsibility of choosing this parameter is in the domain of the algorithm designer.

7.5.1 Convergence

It can be shown that the (noisy) gradient estimate used in the LMS algorithm is an unbiased estimate of the true gradient:

$$
\begin{aligned}
E\left\{\hat{\underline{\nabla}}_k\right\} &= E[-2e(k)\boldsymbol{x}(k)] \\
&= (E[-2(d(k)-\boldsymbol{w}^T(k)\boldsymbol{x}(k))\boldsymbol{x}(k)]) \\
&= 2\boldsymbol{R}\boldsymbol{w}(k)-2\boldsymbol{p} \\
&= \hat{\underline{\nabla}}_k
\end{aligned} \tag{144}
$$

where we have assumed that $\boldsymbol{w}(k)$ and $\boldsymbol{x}(k)$ are statistically independent.

It can be shown that in the mean the LMS will converge to the Wiener-Hopf solution if the step size, μ, is limited by the inverse of the largest eigenvalue. Taking the expectation of both sides of the LMS equation gives:

$$
\begin{aligned}
E\{\boldsymbol{w}(k+1)\} &= E\{\boldsymbol{w}(k)\}+2\mu E[e(k)\boldsymbol{x}(k)] \\
&= E\{\boldsymbol{w}(k)\}+2\mu(E[d(k)\boldsymbol{x}(k)]-E[(\boldsymbol{x}(k)\boldsymbol{x}^T(k))\boldsymbol{w}(k)])
\end{aligned} \tag{145}
$$

and again assuming that $\boldsymbol{w}(k)$ and $\boldsymbol{x}(k)$ are statistically independent:

$$\begin{aligned} E\{\boldsymbol{w}(k+1)\} &= E\{\boldsymbol{w}(k)\} + 2\mu(\boldsymbol{p} - \boldsymbol{R}E\{\boldsymbol{w}(k)\}) \\ &= (\boldsymbol{I} - 2\mu\boldsymbol{R})E\{\boldsymbol{w}(k)\} + 2\mu\boldsymbol{R}\boldsymbol{w}_{opt} \end{aligned} \tag{146}$$

where $\boldsymbol{w}_{opt} = \boldsymbol{R}^{-1}\boldsymbol{p}$ and $\boldsymbol{I}$ is the identity matrix. Now, defining $\boldsymbol{v}(k) = \boldsymbol{w}(k) - \boldsymbol{w}_{opt}$ then we can rewrite the above in the form:

$$E\{\boldsymbol{v}(k+1)\} = (\boldsymbol{I} - 2\mu\boldsymbol{R})E\{\boldsymbol{v}(k)\} \tag{147}$$

For convergence of the LMS to the Wiener-Hopf, we require that $\boldsymbol{w}(k) \to \boldsymbol{w}_{opt}$ as $k \to \infty$, and therefore $\boldsymbol{v}(k) \to \boldsymbol{0}$ as $k \to \infty$. If the eigenvalue decomposition of $\boldsymbol{R}$ is given by $\boldsymbol{Q}^T\Lambda\boldsymbol{Q}$, where $\boldsymbol{Q}^T\boldsymbol{Q} = \boldsymbol{I}$ and Λ is a diagonal matrix then writing the vector $\boldsymbol{v}(k)$ in terms of the linear transformation $\boldsymbol{Q}$, such that $E\{\boldsymbol{v}(k)\} = \boldsymbol{Q}^T E\{\boldsymbol{u}(k)\}$ and multiplying both sides of the above equation, we realizethe decoupled equations:

$$E\{\boldsymbol{u}(k+1)\} = (\boldsymbol{I} - 2\mu\Lambda)E\{\boldsymbol{u}(k)\} \tag{148}$$

and therefore:

$$E\{\boldsymbol{u}(k+1)\} = (\boldsymbol{I} - 2\mu\Lambda)^k E\{\boldsymbol{u}(0)\} \tag{149}$$

where $(\boldsymbol{I} - 2\mu\Lambda)$ is a diagonal matrix:

$$(\boldsymbol{I} - 2\mu\Lambda) = \begin{bmatrix} (1-2\mu\lambda_0) & 0 & 0 & \dots & 0 \\ 0 & (1-2\mu\lambda_1) & 0 & \dots & 0 \\ 0 & 0 & (1-2\mu\lambda_2) & \dots & 0 \\ : & : & : & \dots & 0 \\ 0 & 0 & 0 & 0 & (1-2\mu\lambda_{N-1}) \end{bmatrix} \tag{150}$$

For convergence of this equation to the zero vector, we require that

$$(1-2\mu\lambda_n)^n \to 0 \quad \text{for all} \quad n = 0, 1, 2, \dots, N-1 \tag{151}$$

Therefore the step size, μ, must cater for the largest eigenvalue, $\lambda_{max} = \max(\lambda_0, \lambda_1, \lambda_2, \dots, \lambda_{N-1})$ such that: $|1-2\mu\lambda_{max}| < 1$, and therefore:

$$0 < \mu < \frac{1}{\lambda_{max}} \tag{152}$$

This bound is a necessary and sufficient condition for convergence of the algorithm in the mean square sense. However, this bound is not convenient to calculate, and hence not particularly useful for practical purposes. A more useful sufficient condition for bounding μ can be found using the linear algebraic result that:

$$\text{trace}[\boldsymbol{R}] = \sum_{n=0}^{N-1} \lambda_n \tag{153}$$

i.e. the sum of the diagonal elements of the correlation matrix $\boldsymbol{R}$, is equal to the sum of the eigenvalues, then the inequality:

$$\lambda_{max} \leq \mathrm{trace}[\boldsymbol{R}] \tag{154}$$

will hold. However if the signal $x(k)$ is wide sense stationary, then the diagonal elements of the correlation matrix, $\boldsymbol{R}$, are $E\{x^2(k)\}$ which is a measure of the signal power. Hence:

$$\mathrm{trace}[\boldsymbol{R}] = NE\{x^2(k)\} = N\text{<Signal Power>} \tag{155}$$

and the well known LMS stability bound (sufficient condition) of:

$$0 < \mu < \frac{1}{NE[x_k^2]} \tag{156}$$

is the practical result.

7.5.2 Misadjustment

Misadjustment is a term used in adaptive signal processing to indicate how close the achieved mean squared error (MSE) is to the minimum mean square error. It is defined as the ratio of the excess MSE, to the minimum MSE, and therefore gives a measure of how well the filter can adapt. For the LMS:

$$\begin{aligned}\text{Misadjustment} &= \frac{\text{excess MSE}}{\text{MMSE}} \\ &\approx \mu\text{trace}[\boldsymbol{R}] \\ &\approx \mu N\text{<Signal Power>}\end{aligned} \tag{157}$$

Therefore *misadjustment* from the MMSE solution is proportional to the LMS step size, the filter length, and the signal input power of $x(k)$.

7.5.3 Time Constant

The speed of convergence to a steady state error (expressed as an exponential time constant) can be precisely defined in terms of the eigenvalues of the correlation matrix, $\boldsymbol{R}$. A commonly used (if less accurate) measure is given by:

$$\tau_{mse} \approx \frac{N}{4\mu(\text{trace}[\boldsymbol{R}])} = \frac{1}{4\mu\text{<Signal Power>}} \tag{158}$$

Therefore the speed of adaption is proportional to the inverse of the signal power and the inverse of the step size. A large step size will adapt quickly but with a large MSE, whereas a small step size, will adapt slowly but achieve a small MSE. The design trade-off to select μ, is a requirement of the algorithm designer, and will, of course, depend of the particular application.

7.5.4 Variants

A number of variants of the LMS exist. These variants can be split into three families: (1) algorithms derived to reduce the computation requirements compared to the standard LMS; (2) algorithms derived to improve the convergence properties over the standard LMS; (3) modifications of the LMS to allow a more efficient implementation.

In order to reduce computational requirements, the sign-error, sign-data and sign-sign LMS algorithms circumvent multiplies and replace them with shifting operations (which are essentially power of two multiplies or divides). The relevance of the sign variants of the standard LMS however is now somewhat dated due to the low cost availability of modern DSP processors where a multiply can be performed in the same time as a bit shift (and faster than multiple bit shifts). The convergence speed and achievable mean squared error for all of the sign variants of the LMS are less desirable than the for the standard LMS algorithm.

To improve convergence speed, the stability properties and ensure a small excess mean squared error the normalized, the leaky and the variable step size LMS algorithms have been developed. A summary of some of the LMS variants are:

- **Delay LMS:** The delay LMS simply delays the error signal in order that a "systolic" timed application specific circuit can be implemented:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + 2\mu e(k-n)\mathbf{x}(k-n) \tag{159}$$

  Note that the delay-LMS is in fact a special case of the more general filtered-X LMS.

- **Filtered-X LMS:** See *Least Mean Squares Filtered-X Algorithm*.
- **Filtered-U LMS:** See *Active Noise Control*.
- **Infinite Impulse Response (IIR) LMS:** See *Least Mean Squares - IIR Filter Algorithms*.
- **Leaky LMS:** A leakage factor, c, can be introduced to improve the numerical behavior of the standard LMS:

$$\mathbf{w}(k+1) = c\mathbf{w}(k) + 2\mu e(k)\mathbf{x}(k) \tag{160}$$

  By continually leaking the weight vector, $\mathbf{w}(k)$, even if the algorithm has found the minimum mean squared error solution it will require to continue adapting to compensate for the error introduced by the leakage factor. The advantage of the leakage is that the sensitivity to potentially destabilizing round off errors is reduced. In addition, in applications where the input occasionally becomes very small, *leaky LMS* drives the weights toward zero (this can be an advantage in noise cancelling applications). However the disadvantage to *leaky LMS* is that the achievable mean squared error is not as good as for the standard LMS. Typically c has a value between 0.9 (very leaky) and 1 (no leakage).

- **Newton LMS:** This algorithm improves the convergence properties of the standard LMS. There is quite a high computational overhead to calculate the matrix vector product (and, possibly, the estimate of the correlation matrix $\boldsymbol{R}^{-1}$) at each iteration:

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + 2\boldsymbol{R}^{-1}\mu e(k)\boldsymbol{x}(k) \tag{161}$$

- **Normalised Step Size LMS:** The *normalised LMS* calculates an approximation of the signal input power at each iteration and uses this value to ensure that the step size is appropriate for rapid convergence. The normalized step size, μ_n, is therefore time varying. The normalised LMS is very useful in situations where the input signal power fluctuates rapidly and the input signal is slowly varying non-stationary:

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + 2\mu_n e(k)\boldsymbol{x}(k), \qquad \mu_n = \frac{1}{\varepsilon + \|\boldsymbol{x}(k)\|^2} \tag{162}$$

where ε is a small constant to ensure that in conditions of a zero input signal, $x(k)$, a divide by zero does not occur. $\|\boldsymbol{x}(k)\|$ is 2-norm of the vector $\boldsymbol{x}(k)$.

- **Sign Data/Regressor LMS:** The *sign data (or regressor) LMS* was first developed to reduce the number of multiplications required by the LMS. The step size, μ, is carefully chosen to be a power of two and only bit shifting multiplies are required:

$$\boldsymbol{w}(k+1) = \boldsymbol{w}(k) + 2\mu e(k)\mathrm{sign}[\boldsymbol{x}(k)], \qquad \mathrm{sign}[x(k)] = \begin{cases} 1, & x(k) > 0 \\ 0, & x(k) = 0 \\ -1, & x(k) < 0 \end{cases} \tag{163}$$

- **Sign Error LMS:** The *sign error LMS* was first developed to reduce the number of multiplications required by the LMS. The step size, μ, is carefully chosen to be a power of two and only bit shifting multiplies are required:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + 2\mu \,\mathrm{sign}[e(k)]\mathbf{x}(k), \qquad \mathrm{sign}[e(k)] = \begin{cases} 1, & e(k) > 0 \\ 0, & e(k) = 0 \\ -1, & e(k) < 0 \end{cases} \tag{164}$$

- **Sign-SIgn LMS:** The *sign-sign error LMS* was first presented in 1966 to reduce the number of multiplications required by the LMS. The step size, μ, is carefully chosen to be a power of two and only bit shifting multiplies are required:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + 2\mu \,\mathrm{sign}[e(k)]\mathrm{sign}[\mathbf{x}(k)], \qquad \mathrm{sign}[z(k)] = \begin{cases} 1, & z(k) > 0 \\ 0, & z(k) = 0 \\ -1, & z(k) < 0 \end{cases} \tag{165}$$

- **Variable Step Size LMS:** The variable step size LMS was developed in order that when the LMS algorithm first starts to adapt, the step size is large and convergence is fast. However as the error reduces the step size is automatically decreased in magnitude in order that smaller steps can be taken to ensure that a small excess mean squared error is achieved:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + 2\mu_v e(k)\mathbf{x}(k), \qquad \mu_v \propto E\{e^2(k)\} \tag{166}$$

  Alternatively variable step size algorithms can be set up with deterministic schedules for the modification of the step size. For example

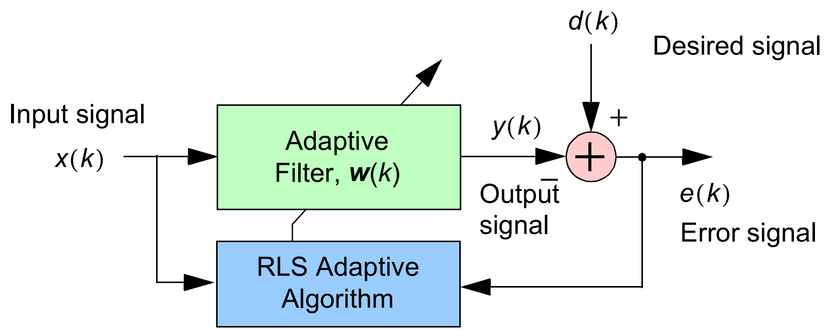
$$\mathbf{w}(k+1) = \mathbf{w}(k) + 2\mu_v e(k)\mathbf{x}(k), \qquad \mu_v = \mu 2^{-\mathrm{int}(\lambda k)} \tag{167}$$

  such that as time, k, passes the step size, μ_v, gets smaller in magnitude. μ is the step size calculated for the standard LMS, λ is a positive constant, and $\mathrm{int}(\lambda k)$ is the closest integer to λk.

Note that a hybrid of more than one of the above LMS algorithm variants could also be implemented.

7.6 Recursive Least Squares (RLS)

The *RLS algorithm* can also be used to update the weights of an adaptive filter where the aim is to minimize the sum of the squared error signal. Consider the adaptive FIR digital filter which is to be updated using an RLS algorithm such that as new data arrives the RLS algorithm uses this new data (innovation) to improve the least squares solution:



$$y(k) = \text{Filter}\{x(k), \mathbf{w}(k)\}$$

$$\mathbf{w}_{k+1} = \mathbf{w}_k + e(k) f\{d((k), x(k))\}$$

For least squares adaptive signal processing the aim is to adapt the impulse response of the FIR digital filter such that the input signal $x(k)$ is filtered to produce $y(k)$ which when subtracted from desired signal $d(k)$, minimises the sum of the squared error signal $e(k)$ over time from the start of the signal at 0 (zero) to the current time k.

Note: While the above figure is reminiscent of the Least Mean Squares (LMS) adaptive filter, the distinction between the two approaches is quite important: LMS minimizes the *mean* of the square of the output error, while RLS minimizes the actual sum of the squared output errors.

In order to minimize the error signal, $e(k)$, consider minimizing the total sum of squared errors for all input signals up to and including time, k. The total squared error, $v(k)$, is:

$$v(k) = \sum_{s=0}^{k} [e(s)]^2 = e^2(0) + e^2(1) + e^2(2) + \ldots + e^2(k) \tag{168}$$

Using vector notation, the error signal can be expressed in a vector format and therefore:

$$\mathbf{e}_k = \begin{bmatrix} e(0) \\ e(1) \\ e(2) \\ : \\ e(k-1) \\ e(k) \end{bmatrix} = \begin{bmatrix} d(0) \\ d(1) \\ d(2) \\ : \\ d(k-1) \\ d(k) \end{bmatrix} - \begin{bmatrix} y(0) \\ y(1) \\ y(2) \\ : \\ y(k-1) \\ y(k) \end{bmatrix} = \mathbf{d}_k - \mathbf{y}_k \tag{169}$$

Noting that the output of the N weight adaptive FIR digital filter is given by:

$$y(k) = \sum_{n=0}^{N-1} w_n x(k-n) = \mathbf{w}^T \mathbf{x}_k = \mathbf{x}_k^T \mathbf{w} \tag{170}$$

where,

$$\mathbf{w} = \begin{bmatrix} w_0 \ w_1 \ w_2 \ \ldots \ w_{N-1} \end{bmatrix}^T \tag{171}$$

and

$$\mathbf{x}_k = [x(k)\ x(k-1)\ x(k-2)\ \dots\ x(k-N+1)]^T \tag{172}$$

then Eq. 169 can be rearranged to give:

$$\mathbf{e}_k = \begin{bmatrix} e(0) \\ e(1) \\ e(2) \\ : \\ e(k-1) \\ e(k) \end{bmatrix} = \mathbf{d}_k - \begin{bmatrix} \mathbf{x}_0^T\mathbf{w} \\ \mathbf{x}_1^T\mathbf{w} \\ \mathbf{x}_2^T\mathbf{w} \\ : \\ \mathbf{x}_{k-1}^T\mathbf{w} \\ \mathbf{x}_k^T\mathbf{w} \end{bmatrix} = \mathbf{d}_k - \begin{bmatrix} \mathbf{x}_0^T \\ \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ : \\ \mathbf{x}_{k-1}^T \\ \mathbf{x}_k^T \end{bmatrix}\mathbf{w}$$

$$= \mathbf{d}_k - \begin{bmatrix} x(0) & 0 & 0 & \dots & 0 \\ x(1) & x(0) & 0 & \dots & 0 \\ x(2) & x(1) & x(0) & \dots & 0 \\ : & : & : & \dots & : \\ x(k-1) & x(k-2) & x(k-3) & \dots & x(k-N) \\ x(k) & x(k-1) & x(k-2) & \dots & x(k-N+1) \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ : \\ w_{N-1} \end{bmatrix} \tag{173}$$

i.e. $\mathbf{e}_k = \mathbf{d}_k - \mathbf{X}_k\mathbf{w}$

where $\boldsymbol{X}_k$ is a $(k+1)\times N$ data matrix made up from input signal samples. Note that the first N rows of $\boldsymbol{X}_k$ are sparse. Eq. 168 can be rewritten such that:

$$\begin{aligned} v(k) &= \mathbf{e}_k^T\mathbf{e}_k = \|\mathbf{e}_k\|_2^2 \\ &= [\boldsymbol{d}_k - \boldsymbol{X}_k\boldsymbol{w}]^T[\boldsymbol{d}_k - \boldsymbol{X}_k\boldsymbol{w}] \\ &= \boldsymbol{d}_k^T\boldsymbol{d}_k + \boldsymbol{w}^T\boldsymbol{X}_k^T\boldsymbol{X}_k\boldsymbol{w} - 2\boldsymbol{d}_k^T\boldsymbol{X}_k\boldsymbol{w} \end{aligned} \tag{174}$$

where $\|\mathbf{e}_k\|_2$ is the 2-norm of the vector $\mathbf{e}_k$. From a first glance at the last line of Eq. 173 it may seem that a viable solution is to set $\boldsymbol{e}_k = \mathbf{0}$ then simply solve the equation $\boldsymbol{w} = \boldsymbol{X}_k^{-1}\boldsymbol{d}_k$. However this is of course not possible in general as $\boldsymbol{X}_k$ is not a square matrix and therefore ***not*** invertible.

In order to find a "good" solution such that the 2-norm of the error vector, $\boldsymbol{e}_k$, is minimized, note that Eq. 174 is quadratic in the vector $\boldsymbol{w}$, and the function $v(k)$ is an up-facing *hyperparaboloid* when plotted in $N+1$ dimensional space, and there exists exactly one minimum point at the bottom of the hyperparaboloid where the gradient vector is zero, i.e.,

$$\frac{\partial}{\partial \boldsymbol{w}}v(k) = \mathbf{0} \tag{175}$$

From Eq. 174

$$\frac{\partial}{\partial \boldsymbol{w}}v(k) = 2\boldsymbol{X}_k^T\boldsymbol{X}_k\boldsymbol{w} - 2\boldsymbol{X}_k^T\boldsymbol{d}_k = -2\boldsymbol{X}_k^T[\boldsymbol{d}_k - \boldsymbol{X}_k\boldsymbol{w}] \tag{176}$$

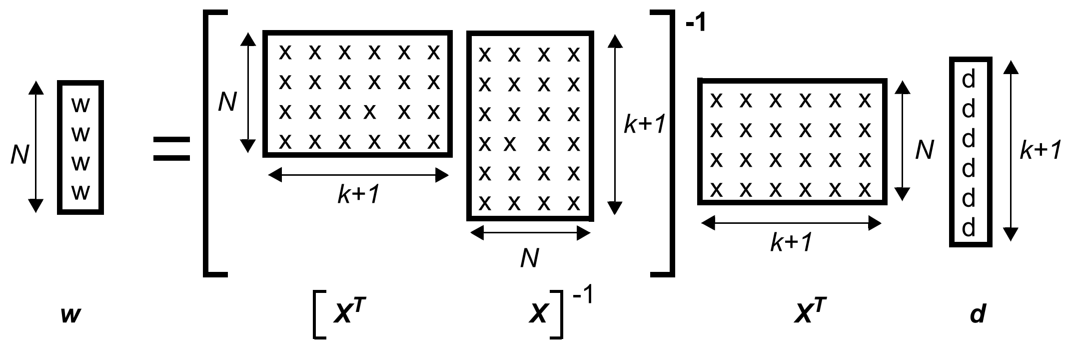
and therefore:

$$
\begin{aligned}
-2\mathbf{X}_k^T[\mathbf{d}_k - \mathbf{X}_k\mathbf{w}_{LS}] &= \mathbf{0} \\
\Rightarrow \mathbf{X}_k^T\mathbf{X}_k\mathbf{w}_{LS} &= \mathbf{X}_k^T\mathbf{d}_k
\end{aligned}
\tag{177}
$$

and the least squares solution, denoted as $\mathbf{w}_{LS}$ and based on data received up to and including time, k, is given as:

$$
\mathbf{w}_{LS} = [\mathbf{X}_k^T\mathbf{X}_k]^{-1}\mathbf{X}_k^T\mathbf{d}_k \tag{178}
$$

Note that because $[\mathbf{X}_k^T\mathbf{X}_k]$ is a symmetric square matrix, then $[\mathbf{X}_k^T\mathbf{X}_k]^{-1}$ is also a symmetric square matrix. As with any linear algebraic manipulation a useful check is to confirm that the matrix dimensions are compatible, thus ensuring that $\mathbf{w}_{LS}$ is a $N \times 1$ matrix:

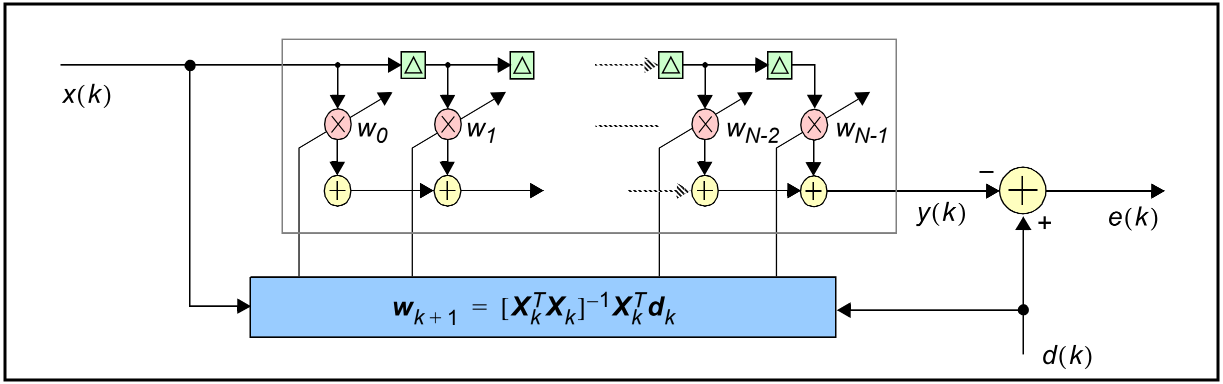


Note that if in the special case where $\mathbf{X}_k$ is a square non-singular matrix, then Eq. 178 simplifies to:

$$
\mathbf{w}_{LS} = \mathbf{X}_k^{-1}\mathbf{X}_k^{-T}\mathbf{X}_k^T\mathbf{d}_k = \mathbf{X}_k^{-1}\mathbf{d}_k \tag{179}
$$

The computation to calculate Eq. 178 requires about $O(N^4)$ MACs (multiply/accumulates) and $O(N)$ divides for the matrix inversion, and $O((k+1)\times N^2)$ MACs for the matrix multiplications. Clearly therefore, the more data that is available, then the more computation required.

At time iteration $k+1$, the weight vector to use in the adaptive FIR filter that minimizes the 2-norm of the error vector, $\mathbf{e}_k$ can be denoted as $\mathbf{w}_{k+1}$, and the open loop least squares adaptive filter solution can be represented as the block diagram:



Note however that at time $k+1$ when a new data sample arrives at both the input, $x(k+1)$, and the desired input, $d(k+1)$ then this new information should ideally be incorporated in the least squares solution with a view to obtaining an improved solution. The new least squares filter weight vector to use at time $k+2$ (denoted as $\mathbf{w}_{k+2}$) is clearly given by:

$$\mathbf{w}_{k+2} = [\mathbf{X}_{k+1}^T\mathbf{X}_{k+1}]^{-1}\mathbf{X}_{k+1}^T\mathbf{d}_{k+1} \tag{180}$$

This equation requires that another full matrix inversion is performed, $[\mathbf{X}_{k+1}^T\mathbf{X}_{k+1}]^{-1}$, followed by the appropriate matrix multiplications. This very high level of computation for every new data sample provides the motivation for deriving the *recursive least squares* (RLS) algorithm. RLS has a much lower level of computation by calculating $\mathbf{w}_{k+1}$ using the result of previous estimate $\mathbf{w}_k$ to reduce computation.

Consider the situation where we have calculated $\mathbf{w}_k$ from

$$\mathbf{w}_k = [\mathbf{X}_{k-1}^T\mathbf{X}_{k-1}]^{-1}\mathbf{X}_{k-1}^T\mathbf{d}_{k-1} = \mathbf{P}_{k-1}\mathbf{X}_{k-1}^T\mathbf{d}_{k-1} \tag{181}$$

where

$$\mathbf{P}_{k-1} = [\mathbf{X}_{k-1}^T\mathbf{X}_{k-1}]^{-1} \tag{182}$$

When the new data samples, $x(k)$ and $d(k)$, arrive we have to calculate:

$$\mathbf{w}_{k+1} = [\mathbf{X}_k^T\mathbf{X}_k]^{-1}\mathbf{X}_k^T\mathbf{d}_k = \mathbf{P}_k\mathbf{X}_k^T\mathbf{d}_k \tag{183}$$

However note that $\mathbf{P}_k$ can be written in terms of the previous data matrix $\mathbf{X}_{k-1}$ and the data vector $\mathbf{x}_k$ by partitioning the matrix $\mathbf{X}_k$:

$$\begin{aligned}\mathbf{P}_k = [\mathbf{X}_k^T\mathbf{X}_k]^{-1} &= \begin{bmatrix}\mathbf{X}_{k-1}^T & \mathbf{x}_k\end{bmatrix}\begin{bmatrix}\mathbf{X}_{k-1} \\ \mathbf{x}_k^T\end{bmatrix} \\ &= \left[\mathbf{X}_{k-1}^T\mathbf{X}_{k-1} + \mathbf{x}_k\mathbf{x}_k^T\right]^{-1} \\ &= \left[\mathbf{P}_{k-1}^{-1} + \mathbf{x}_k\mathbf{x}_k^T\right]^{-1}\end{aligned} \tag{184}$$

where, of course, $\boldsymbol{x}_k = [x(k+1), x(k), x(k-1), \ldots, x(k-N+1)]$ as before in Eq. 170. In order to write Eq. 184 in a more "suitable form" we use the matrix inversion lemma which states that:

$$[\boldsymbol{A}^{-1} + \boldsymbol{BCD}]^{-1} = \boldsymbol{A} - \boldsymbol{AB}[\boldsymbol{C} + \boldsymbol{DAB}]^{-1}\boldsymbol{DA} \tag{185}$$

where $\boldsymbol{A}$ is a non-singular matrix and $\boldsymbol{B}$, $\boldsymbol{C}$ and $\boldsymbol{D}$ are appropriately dimensioned matrices. Using the matrix inversion lemma of Eq.184, where $\boldsymbol{P}_{k-1} = \boldsymbol{A}$, $\boldsymbol{x}_k = \boldsymbol{B}$, $\boldsymbol{x}_k^T = \boldsymbol{D}$ and $\boldsymbol{C}$ is the 1×1 identity matrix. i.e. the scalar 1, then:

$$\boldsymbol{P}_k = \boldsymbol{P}_{k-1} - \boldsymbol{P}_{k-1}\boldsymbol{x}_k[1 + \boldsymbol{x}_k^T\boldsymbol{P}_{k-1}\boldsymbol{x}_k]^{-1}\boldsymbol{x}_k^T\boldsymbol{P}_{k-1} \tag{186}$$

This equation implies that if we know the matrix $[\boldsymbol{X}_{k-1}^T\boldsymbol{X}_{k-1}]^{-1}$ then the matrix $[\boldsymbol{X}_k^T\boldsymbol{X}_k]^{-1}$ can be computed without explicitly performing a complete matrix inversion from first principles. This, of course, saves in computation effort. Eq. 183 and 186 are one form of the RLS algorithm. By additional algebraic manipulation, the computation complexity of Eq. 186 can be simplified even further.

By substituting Eq. 186 into Eq. 183, and partitioning the vector $\boldsymbol{d}_k$ and simplifying gives:

$$\begin{aligned}
\boldsymbol{w}_{k+1} &= [\boldsymbol{P}_{k-1} - \boldsymbol{P}_{k-1}\boldsymbol{x}_k[1 + \boldsymbol{x}_k^T\boldsymbol{P}_{k-1}\boldsymbol{x}_k]^{-1}\boldsymbol{x}_k^T\boldsymbol{P}_{k-1}]\boldsymbol{X}_k^T\boldsymbol{d}_k \\
&= [\boldsymbol{P}_{k-1} - \boldsymbol{P}_{k-1}\boldsymbol{x}_k[1 + \boldsymbol{x}_k^T\boldsymbol{P}_{k-1}\boldsymbol{x}_k]^{-1}\boldsymbol{x}_k^T\boldsymbol{P}_{k-1}]\begin{bmatrix}\boldsymbol{X}_{k-1}^T & \boldsymbol{x}_k\end{bmatrix}\begin{bmatrix}\boldsymbol{d}_{k-1} \\ d(k)\end{bmatrix} \\
&= [\boldsymbol{P}_{k-1} - \boldsymbol{P}_{k-1}\boldsymbol{x}_k[1 + \boldsymbol{x}_k^T\boldsymbol{P}_{k-1}\boldsymbol{x}_k]^{-1}\boldsymbol{x}_k^T\boldsymbol{P}_{k-1}]\begin{bmatrix}\boldsymbol{X}_{k-1}^T\boldsymbol{d}_{k-1} + \boldsymbol{x}_k d(k)\end{bmatrix}
\end{aligned} \tag{187}$$

Using the substitution $\boldsymbol{w}_k = \boldsymbol{P}_{k-1}\boldsymbol{X}_{k-1}^T\boldsymbol{d}_{k-1}$ and also dropping the time subscripts for notational convenience, i.e. $\boldsymbol{P} = \boldsymbol{P}_{k-1}$, $\boldsymbol{x} = \boldsymbol{x}_k$, $\boldsymbol{d} = \boldsymbol{d}_{k-1}$, and $d = d(k)$), further simplification can be performed:

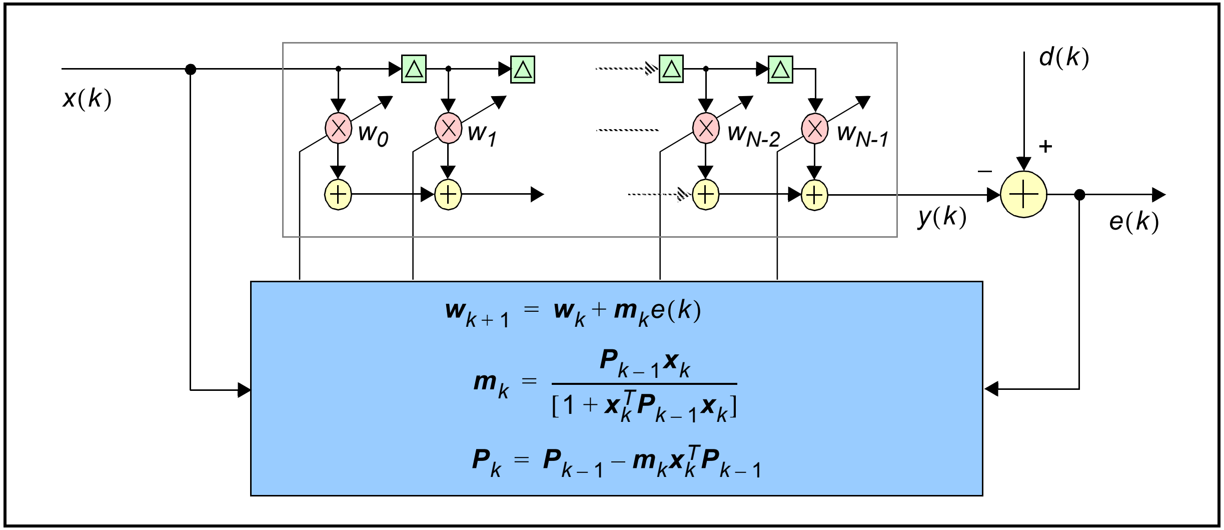
$$\begin{aligned}
\boldsymbol{w}_{k+1} &= [\boldsymbol{P} - \boldsymbol{P}\boldsymbol{x}[1 + \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}]^{-1}\boldsymbol{x}^T\boldsymbol{P}]\left[\boldsymbol{X}^T\boldsymbol{d} + \boldsymbol{x}d\right] \\
&= \boldsymbol{P}\boldsymbol{X}^T\boldsymbol{d} + \boldsymbol{P}\boldsymbol{x}d - \boldsymbol{P}\boldsymbol{x}[1 + \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}]^{-1}\boldsymbol{x}^T\boldsymbol{P}\boldsymbol{X}^T\boldsymbol{d} - \boldsymbol{P}\boldsymbol{x}[1 + \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}]^{-1}\boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}d \\
&= \boldsymbol{w}_k - \boldsymbol{P}\boldsymbol{x}[1 + \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}]^{-1}\boldsymbol{x}^T\boldsymbol{w}_k + \boldsymbol{P}\boldsymbol{x}d - \boldsymbol{P}\boldsymbol{x}[1 + \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}]^{-1}\boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}d \\
&= \boldsymbol{w}_k - \boldsymbol{P}\boldsymbol{x}[1 + \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}]^{-1}\boldsymbol{x}^T\boldsymbol{w}_k + \boldsymbol{P}\boldsymbol{x}d[1 - [1 + \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}]^{-1}\boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}] \\
&= \boldsymbol{w}_k - \boldsymbol{P}\boldsymbol{x}[1 + \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}]^{-1}\boldsymbol{x}^T\boldsymbol{w}_k + \boldsymbol{P}\boldsymbol{x}[1 + \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}]^{-1}[[1 + \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}] - \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}]d \\
&= \boldsymbol{w}_k - \boldsymbol{P}\boldsymbol{x}[1 + \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}]^{-1}\boldsymbol{x}^T\boldsymbol{w}_k + \boldsymbol{P}\boldsymbol{x}[1 + \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}]^{-1}d \\
&= \boldsymbol{w}_k + \boldsymbol{P}\boldsymbol{x}[1 + \boldsymbol{x}^T\boldsymbol{P}\boldsymbol{x}]^{-1}(d - \boldsymbol{x}^T\boldsymbol{w}_k)
\end{aligned} \tag{188}$$

and reintroducing the subscripts, and noting that $y(k) = \boldsymbol{x}_k^T\boldsymbol{w}_k$:

$$\begin{aligned}
\boldsymbol{w}_{k+1} &= \boldsymbol{w}_k + \boldsymbol{P}_{k-1}\boldsymbol{x}_k[1 + \boldsymbol{x}_k^T\boldsymbol{P}_{k-1}\boldsymbol{x}_k]^{-1}(d(k) - y(k)) \\
&= \boldsymbol{w}_k + \boldsymbol{m}_k(d(k) - y(k)) \\
&= \boldsymbol{w}_k + \boldsymbol{m}_k e(k)
\end{aligned} \tag{189}$$

where $\boldsymbol{m}_k = \boldsymbol{P}_{k-1}\boldsymbol{x}_k[1 + \boldsymbol{x}_k^T\boldsymbol{P}_{k-1}\boldsymbol{x}_k]^{-1}$ and is called the gain vector.

The RLS adaptive filtering algorithm therefore requires that at each time step, the vector $\boldsymbol{m}_k$ and the matrix $\boldsymbol{P}_k$ are computed. The filter weights are then updated using the error output, $e(k)$. Therefore the block diagram for the closed loop RLS adaptive FIR filter is: :



The above form of the RLS requires $O(N^2)$ MACs and one divide on each iteration.

7.6.1 Exponentially Weighted

One problem with least squares and *recursive least squares (RLS)* algorithm derived in entry *Recursive Least Squares*, is that the minimization of the 2-norm of the error vector $\boldsymbol{e}_k$ calculates the least squares vector at time k based on all previous data, i.e. data from long ago is given as much relevance as recently received data. Therefore if at some time in the past a block of “bad” data was received or the input signal statistics changed then the RLS algorithm will calculate the current least squares solution giving as much relevance to the old (and probably irrelevant) data as it does to very recent inputs. Therefore the RLS algorithm has infinite memory.

In order to overcome the infinite memory problem, the exponentially weighted least squares, and exponentially weighted recursive least squares (EW-RLS) algorithms can be derived. Consider again Eq. 168 where this time each error sample is weighted using a forgetting factor constant λ which just less than 1:

$$v(k) = \sum_{s=0}^{k} \lambda^{k-s}[e(s)]^2 = \lambda^k e^2(0) + \lambda^{k-1} e^2(1) + \lambda^{2k-2} e^2(2) + \ldots + e^2(k) \tag{190}$$

For example if a forgetting factor of 0.9 was chosen then data which is 100 time iterations old is pre-multiplied by $0.9^{100} = 2.6561 \times 10^{-5}$ and thus considerably de-emphasized compared to the current data. Therefore in dB terms, data that is more 100 time iterations old is attenuated by $10\log(0.00026561) = -46\text{ dB}$. Data that is more than 200 time iterations old is therefore attenuated by around 92 dB, and if the input data were 16 bit fixed point corresponding to a dynamic range of 96 dB, then the old data is on the verge of being completely forgotten about. The forgetting factor is typically a value of between 0.9 and 0.9999.

Noting the form of Eq. 174 we can rewrite Eq. 190 as:

$$v(k) = \mathbf{e}_k^T \Lambda_k \mathbf{e}_k \tag{191}$$

where Λ_k is a $(k+1) \times (k+1)$ diagonal matrix $\Lambda_k = \mathrm{diag}[\lambda^k, \lambda^{k-1}, \lambda^{k-2}, \ldots, \lambda, 1]$

Therefore:

$$
\begin{aligned}
v(k) &= [\mathbf{d}_k - \mathbf{X}_k\mathbf{w}]^T\Lambda_k[\mathbf{d}_k - \mathbf{X}_k\mathbf{w}] \\
&= \mathbf{d}_k^T\Lambda_k\mathbf{d}_k + \mathbf{w}^T\mathbf{X}_k^T\Lambda_k\mathbf{X}_k\mathbf{w} - 2\mathbf{d}_k^T\Lambda_k\mathbf{X}_k\mathbf{w}
\end{aligned} \tag{192}
$$

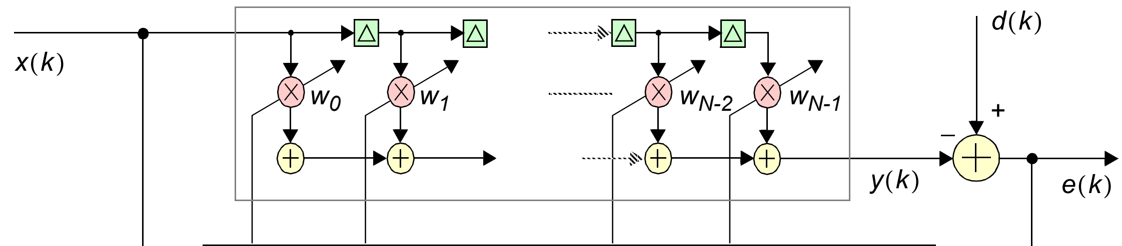
Following the same procedure as for Eqs. 175 to 178 the exponentially weight least squares solution is easily found to be:

$$
\mathbf{w}_{LS} = [\mathbf{X}_k^T\Lambda_k\mathbf{X}_k]^{-1}\mathbf{X}_k^T\Lambda_k\mathbf{d}_k \tag{193}
$$

In the same way as the RLS algorithm was realised, we can follow the same approach as Eqs. 181 to 189 and realizethe exponentially weighted RLS algorithm:

$$
\begin{aligned}
\mathbf{w}_{k+1} &= \mathbf{w}_k + \mathbf{m}_k e(k) \\
\mathbf{m}_k &= \frac{\mathbf{P}_{k-1}\mathbf{x}_k}{[\lambda + \mathbf{x}_k^T\mathbf{P}_{k-1}\mathbf{x}_k]} \\
\mathbf{P}_k &= \frac{\mathbf{P}_{k-1} - \mathbf{m}_k\mathbf{x}_k^T\mathbf{P}_{k-1}}{\lambda}
\end{aligned} \tag{194}
$$

Therefore the block diagram for the exponentially weighted RLS algorithm is:



$$\mathbf{w}_{k+1} = \mathbf{w}_k + \mathbf{m}_k e(k)$$

$$\mathbf{m}_k = \frac{\mathbf{P}_{k-1}\mathbf{x}_k}{[\lambda + \mathbf{x}_k^T \mathbf{P}_{k-1}\mathbf{x}_k]}$$

$$\mathbf{P}_k = \frac{\mathbf{P}_{k-1} - \mathbf{m}_k \mathbf{x}_k^T \mathbf{P}_{k-1}}{\lambda}$$

Compared to the Least Mean Squares (LMS) algorithm, the RLS can provide much faster convergence and a smaller error, however the computation required is a factor of N more than for the LMS, where N is the adaptive filter length. The RLS is less numerically robust than the LMS.