CHEBYSHEV LOW PASS FIR FILTER DESIGN FOR REDUCTION OF NOISE IN HUMAN SPEECH

By

Kundan Singh 031031

Kshitij Sharma 031110

Puneet Jain 031068

विद्या तत्व ज्योतिसम:

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

May-2007

Submitted in partial fulfillment of the Degree of Bachelor of Technology

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

CERTIFICATE

This is to certify that the work entitled, "Chebyshev Low Pass Fir Filter Design For Reduction Of Noise In Human Speech" submitted by Kundan Singh, Kshitij Sharma and Puneet Jain in partial fulfillment for the award of degree of Bachelor of Technology in Electronics and Communication Engineering of Jaypee University of Information Technology has been carried out under my supervision. This work has not been submitted partially or wholly to any other University or Institute for the award of this or any other degree or diploma.

[illegible]

Mr. Vinay Kumar
Lecturer ,
Department of Electronics and Communication Engineering
Jaypee University Of Information Technology.

[illegible]

Prof. Sunil Bhooshan
H.O.D. ,
Department Of Electronics and Communication Engineering
Jaypee University Of Information Technology.

ACKNOWLEDGEMENT

In Accordance with our final project submission of 8th Semester(B.Tech Electronics and Communication Engg.), we were assigned to study and research on Chebyshev Low Pass FIR Filter Design For Reduction of Noise In Human Speech and making an application for same. We would like to express our extreme gratitude to

Mr. Vinay Kumar
Lecturer ,
Department of Electronics and Communication Engineering
Jaypee University Of Information Technology.

Prof. Sunil Bhooshan
H.O.D. ,
Department Of Electronics And Communication Engineering
Jaypee University Of Information Technology.

for guiding us, being extremely helpful, patient and always being there whenever we were in doubt. Without his help, support and constant supervision, we would have never been able to complete our final project assignment successfully.

CONTENTS

LIST OF FIGURES

LIST OF EQUATIONS

ABSTRACT

Speech filtering is one of the classical topics of speech signal processing and several methods have been suggested as well as employed. The filters used for this are mainly divided into two categories viz. analog filters and digital Filters. The digital filters have many advantages over traditional analog filters such as accuracy, ease of design. For speech filtering FIR is most commonly used due to the advantages such as linear phase and guaranteed stability.

In this paper we will use a Chebyshev FIR low pass filter based on antenna theory approach to filter noise from speech signal. Low-pass filters allow the low frequency components of an input signal to pass through while attenuating high frequency components. Measurement noise falls into the high frequency range of the signal spectrum, while the underlying process signal usually lies towards the low frequency end. Thus filters that are used to remove noise from measurements are of the low-pass types. A signal can be decomposed into components of different frequencies. Thus, one method of examining the capabilities of filters is to look at what happens to inputs of different frequencies when they are passed through the filter.

1 STUDY AND RESEARCH

1.1 INTRODUCTION TO FILTER DESIGNING

Filter design is the process of working out a filter (in the sense in which the term is used in signal processing, statistics, and applied mathematics), often a linear shift-invariant filter, which satisfies a set of requirements, some of which are contradicting. The problem to be solved is to find a realization of the filter which met each of the requirements to a sufficient degree to make it useful.

The filter design process can be described as an optimization problem where each requirement contributes with a term to an error function which should be minimized. Certain parts of the design process can be automated, but normally it is necessary to use an experienced electrical engineer to get a good result.

1.1.1 Typical design requirements

Typical requirements which are considered in the design process are

- The filter should have a specific frequency function
- The filter should have a specific impulse response
- The filter should be causal
- The filter should be stable
- The filter should be localized
- The computational complexity of the filter should be low
- The filter should be implemented in a particular hardware or software.

The frequency function

Typical examples of frequency function are

- A low-pass filter is used to block unwanted high-frequency signals.
- A high-pass filter passes high frequencies fairly well; it is helpful as a filter to block any unwanted low frequency components.
- A band-pass filter passes a limited range of frequencies.

- A band-stop filter passes frequencies above or below a certain range. This is the least common filter.
- A low-shelf filter passes all frequencies, but boosts or cuts frequencies below the cutoff frequency by specified amount.
- A high-shelf filter passes all frequencies, but boosts or cuts frequencies above the cutoff frequency by specified amount.
- A peak EQ filter makes a peak or a dip in the frequency response, commonly used in graphic equalizers.
- An all-pass filter passes through all frequencies unchanged, but changes the phase of the signal. This is a filter commonly used in phaser effects.

An important parameter is the required frequency response. In particular, the steepness and complexity of the response curve is a deciding factor for the filter order and feasibility.

A first order filter will only have a single frequency-dependent component. This means that the slope of the frequency response is limited to 6 dB per octave. For many purposes, this is not sufficient. To achieve steeper slopes, higher order filters are required.

I relation to the desired frequency function, there may also be an accompanying weighting function which describes, for each frequency, how important it is that the resulting frequency function approximates the desired one. The larger weight, the more important is a close approximation.

The impulse response

There is a direct correspondence between the filter's frequency function and its impulse response, the former is the Fourier transform of the latter. This means that any requirement on the frequency function is a requirement on the impulse response, and vice versa. However, in certain applications it may be the filter's impulse response which is explicit and the design process then aims at producing an as close approximation as possible to the requested impulse response given all other requirements.

Causality

In order to be implementable, any time-dependent filter must be causal: the filter response only depends on the current and past inputs. A standard approach is to leave this requirement until the final step. If the resulting filter is not causal, it can be made causal by introducing an appropriate time-shift (or delay). If the filter is a part of a larger system (which it normally is) these types of delays have to be introduced with care since they affect the operation of the entire system.

Stability

A stable filter assures that every limited input signal produces a limited filter response. A filter which does not meet this requirement may in some situations prove useless or even harmful. Certain design approaches can guarantee stability, for example by using only feed-forward circuits such as an FIR filter. On the other hand, filter based on feedback circuits have other advantages and may therefore be preferred, even if this class of filters include unstable filters. In this case, the filters must be carefully designed in order to avoid instability.

Locality

In certain applications we have to deal with signals which contain components which can be described as local phenomena, for example pulses or steps, which have certain time duration. A consequence of applying a filter to a signal is, in intuitive terms, that the duration of the local phenomena is extended by the width of the filter. This implies that it is sometimes important to keep the filter width as short as possible.

Computational complexity

A general desire in any design is that the number of operations (additions and multiplications) needed to compute the filter response is as low as possible. In certain applications, this desire is a strict requirement, for example due to limited computational resources, limited power resources, or limited time. The last limitation is typical in real-time applications. There are several ways in which a filter can have different computational complexity. For example, the order of a filter is more or less proportional to the number of operations. This means that by choosing a low order filter, the computation time can be reduced. For discrete filters the computational complexity is more or less proportional to the number of filter coefficients. If the filter has many coefficients, for example in the case of multidimensional signals such as tomography data, it may be relevant to reduce the number of coefficients by removing those which are sufficiently close to zero.

Other considerations

It must also be decided how the filter is going to be implemented:

- Analog filter
- Analog sampled filter
- Digital filter
- Mechanical filter

1.2 Digital Filter

In electronics, a digital filter is any electronic filter that works by performing digital mathematical operations on an intermediate form of a signal. This is in contrast to older analog filters which work entirely in the analog realm and must rely on physical networks of electronic components (such as resistors, capacitors, transistors, etc.) to achieve the desired filtering effect.

Digital filters can achieve virtually any filtering effect that can be expressed as a mathematical function or algorithm. The two primary limitations of digital filters are their speed (the filter can't operate any faster than the computer at the heart of the filter), and their cost. However as the cost of integrated circuits has continued to drop over time, digital filters have become increasingly commonplace and are now an essential element of many everyday objects such as radios, cellphones, and stereo receivers.

1.2.1 Digital filter advantages

Digital filters can easily realize performance characteristics far beyond what are implementable with analog filters. It is not particularly difficult, for example, to create a 1000 Hz low-pass filter which can achieve near-perfect transmission of a 999 Hz input while entirely blocking a 1001 Hz signal. Analog filters cannot discriminate between such closely spaced signals.

Also, for complex multi-stage filtering operations, digital filters have the potential to attain much better signal to noise ratios than analog filters. This is because whereas at each intermediate stage the analog filter adds more noise to the signal, the digital filter performs noiseless mathematical operations at each intermediate step in the transform. The primary source of noise in a digital filter is to be found in the initial ADC - analog to digital conversion step, where in addition to any circuit noise introduced, the signal is subject to an unavoidable quantization error which is due to the finite resolution of the digital representation of the signal.

1.2.2 Types of digital filters

Digital filters are implemented according to one of two basic principles, according to how they respond to an impulse:

Infinite impulse response (IIR)
Finite impulse response (FIR)

Many digital filters are based on the Fast Fourier transform, a mathematical algorithm that quickly extracts the frequency spectrum of a signal, allowing the spectrum to be manipulated (such as to create pass-band filters) before converting the modified spectrum back into a time-series signal.

Another form of a typical linear digital filter, expressed as a transform in the Z-domain, is

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + a_2 z^{-2} + \cdots + a_M z^{-M}} \qquad (1.2.1)$$

where M is the order of the filter.

Infinite Impulse Response

IIR filters are the digital counterpart to analog filters. They use feedback, and will normally require less computing resources than an FIR filter of similar performance. Due to the feedback, high order IIR filters may have problems with instability and arithmetic overflow, and require careful design to avoid such pitfalls. Additionally, they have an inherent frequency-dependent phase shift, which can be a problem in many situations. 2nd order IIR filters are often called 'biquads' and a common implementation of higher order filters is to cascade biquads.

Finite impulse response

A finite impulse response (FIR) filter is a type of a digital filter. It is 'finite' because its response to an impulse ultimately settles to zero. This is in contrast to infinite impulse response filters which have internal feedback and may continue to respond indefinitely.

Properties

- A FIR filter has a number of useful properties which sometimes make it preferable to an infinite impulse response filter. FIR filters:
- Are inherently stable. This is due to the fact that all the poles are located at the origin and thus are located within the unit circle.
- Require no feedback. This means that any rounding errors are not compounded by summed iterations. The same relative error occurs in each calculation.
- Can have linear phase
- Can have minimum phase

The general form of the difference equation for a linear, time-invariant, discrete-time system (LTIDT system) is

$$y(n) = -\sum_{k=1}^{N} a(k)y(n-k) + \sum_{k=0}^{M} b(k)x(n-k) \tag{1.2.2}$$

The transfer function for such a system is given by

$$H(z^{-1}) = \frac{b_0 + b(1)z^{-1} + b(2)z^{-2} + \cdots + b(M)z^{-M}}{1 + a(1)z^{-1} + a(2)z^{-2} + a(3)z^{-3} + \cdots + a(N)z^{-N}} \tag{1.2.3}$$

The transfer function of an FIR filter, in particular, is given by

$$H(z^{-1}) = b_0 + b(1)z^{-1} + b(2)z^{-2} + \cdots + b(M)z^{-M} \tag{1.2.4}$$

and the difference equation describing this FIR filter is given by

$$\begin{aligned} y(n) &= \sum_{k=0}^{M} b(k)x(n-k) \\ &= b(0)x(n) + b(1)x(n-1) + \cdots + b(M)x(n-M) \end{aligned} \tag{1.2.5}$$

We get the output y(n) due to the unit sample input δ(n) to be exactly the values b(0), b(1), b(2), b(3), . . . , b(M). The output due to the unit sample function δ(n) is the unit sample response or the unit impulse response denoted by h(n). So the samples of the unit impulse response h(n) = b(n), which means that the unit impulse response h(n) of the discrete-time system described by the difference equation is finite in length. That is why the system is called the finite impulse response filter or the FIR filter. It has also been known by other names such as the transversal filter, nonrecursive filter, moving-average filter, and tapped delay filter. Since h(n) = b(n) in the case of an FIR filter, we can represent

$$H(z^{-1}) = \sum_{k=0}^{M} h(k)z^{-k} = h(0) + h(1)z^{-1} + h(2)z^{-2} + \cdots + h(M)z^{-(M)} \qquad (1.2.6)$$

The FIR filters have a few advantages over the IIR filters

We can easily design the FIR filter to meet the required magnitude response in such a way that it achieves a constant group delay. *Group delay* is defined as $\tau = -(d\theta/d\omega)$, where θ is the phase response of the filter. The phase response of a filter with a constant group delay is therefore a linear function of frequency. It transmits all frequencies with the same amount of delay, which means that there will not be any phase distortion and the input signal will be delayed by a constant when it is transmitted to the output. A filter with a constant group delay is highly desirable in the transmission of digital signals.

The samples of its unit impulse response are the same as the coefficients of the transfer function. There is no need to calculate *h(n)* from *H(z−1)*, such as during every stage of the iterative optimization procedure or for designing the structures (circuits) from *H(z−1)*.The FIR filters are always stable and are free from limit cycles that arise as a result of finite word length representation of multiplier constants and signal values. The effect of finite word length on the specified frequency response or the time-domain response or the output noise is smaller than that for IIR filters.

Although the unit impulse response *h(n)* of an IIR filter is an infinitely long sequence, it is reasonable to assume in most practical cases that the value of the samples becomes almost negligible after a finite number; thus, choosing a sequence of finite length for the discrete-time signal allows us to use powerful numerical methods for processing signals of finite length.

1.3 LINEAR ARRAY OF n ELEMENTS

1.3.1 Linear Arrays Of n Isotropic Point Sources

Taking the case of a linear equispaced antenna array with n elements of equal amplitude and spacing, labelled from left to right where n is a positive integer. The total field 'E' at a large distance in the direction □ is given as

$$E = 1 + e^{j\psi} + e^{j2\psi} + e^{j3\psi} + \dots\dots + e^{j(n-2)\psi} + e^{j(n-1)\psi} \qquad (1.3.1)$$

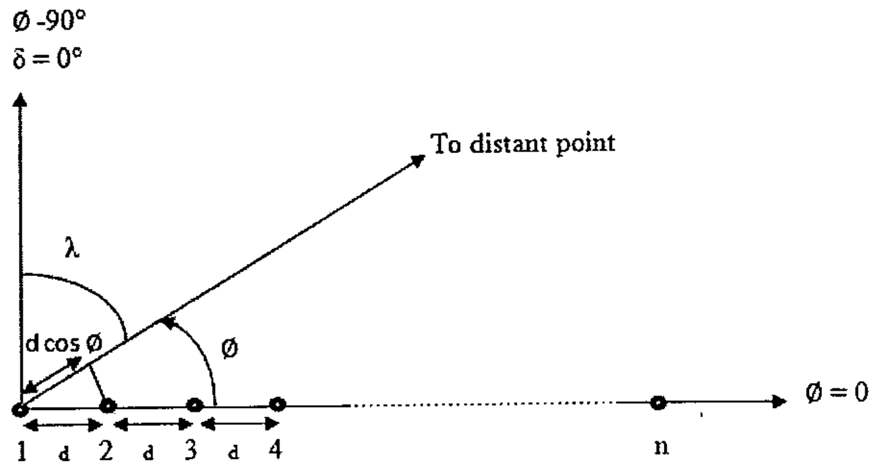


Figure 1 : Linear Arrays Of n Isotropic Point Source

where ψ the total phase difference of the fields from adjacent sources as given by

$$\psi = \frac{2\pi d}{\lambda} \cos \phi + \delta = kd \cos \phi + \delta \qquad (1.3.2)$$

where $k = \frac{2\pi}{\lambda}$

where δ is the phase difference of adjacent sources, i.e. source 2 with respect to 1,3 with respect to 2 etc.

Taking the case of a linear equispaced antenna array with n elements with different amplitude, labelled from left to right.

$$|E| = |A_0 e^{j0} + A_1 e^{j\psi} + A_2 e^{j2\psi} + \dots\dots\dots + A_{n-2} e^{j(n-2)\psi} + A_{n-1} e^{j(n-1)\psi}| \qquad (1.3.3)$$

where ψ the total phase difference of the fields from adjacent sources as given by

$$\psi = \frac{2\pi d}{\lambda} \cos ø + \delta = kd \cos ø + \delta$$

Where $|E|$ is the magnitude of the far field

$k = 2\pi / \lambda$

λ is the free space wavelength,

d is the spacing between elements,

$ø$ is the angle from the normal to the linear array,

δ is the progressive phase shift from left to right,

and $A_0, A_1, A_2, \dots..$ are complex amplitudes which are proportional to the current amplitudes.

Suppose that there are N isotropic radiators equally spaced along the z-axis and fed with equal amplitudes. We assign a fixed phase shift δ between progressive elements. The array factor field is

$$\frac{\sin(N\psi/2)}{N \sin(\psi/2)} \qquad (1.3.4)$$

where $\psi = kd \cos ø + \delta$

We use this to plot a universal radiation pattern for the array for two to 10 elements. The abscissa ψ is plotted in degrees (360° is substituted for 2π in k). Both ends of the plot are lines of symmetry. The plot is periodic (period 360°). We see that the level of the first sidelobe ($N = 2$ has no sidelobe) decreases as N decreases but approaches a limit of 13.3 dB of the continuous aperture.

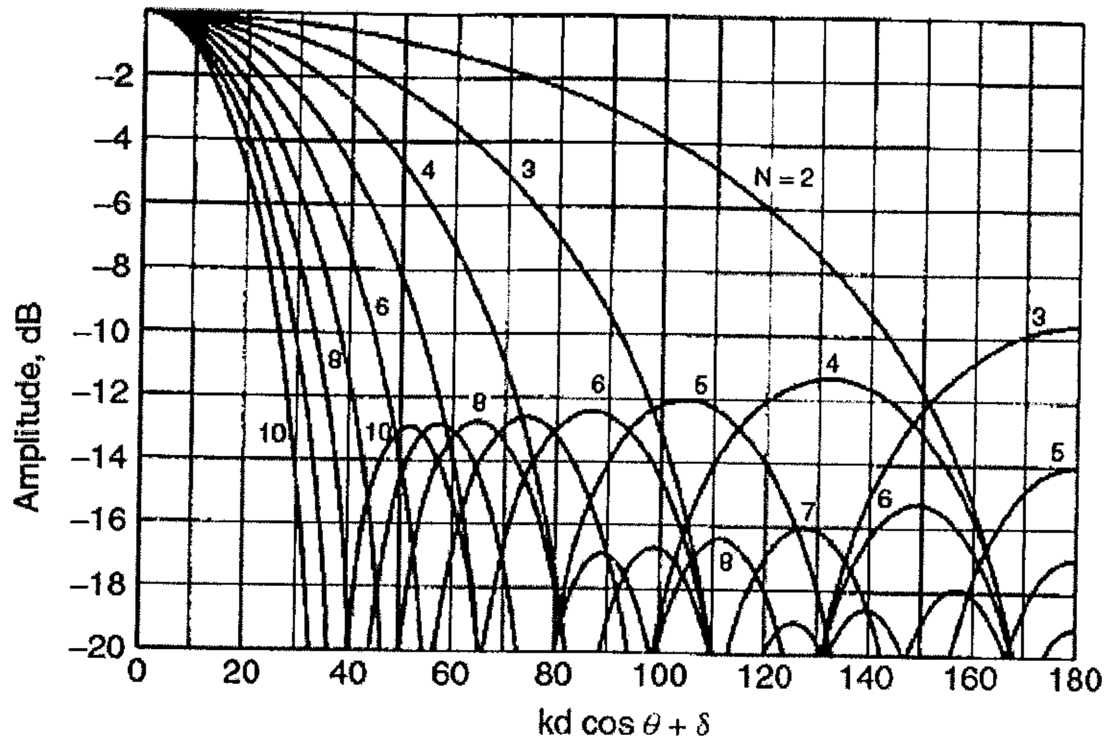


Figure 2 : ψ-space pattern of linear arrays with a uniform amplitude distribution.

Above figure demonstrates the periodic pattern for $N = 6$ and shows a projection to a polar pattern when the progressive phase between elements is zero and the elements are spaced $\lambda/2$. We can plot similar curves for other array distributions; all have a period of 360°. An array can be analyzed as a sampling of the continuous distribution that produces a Fourier series of the distribution. A Fourier series has multiple responses.

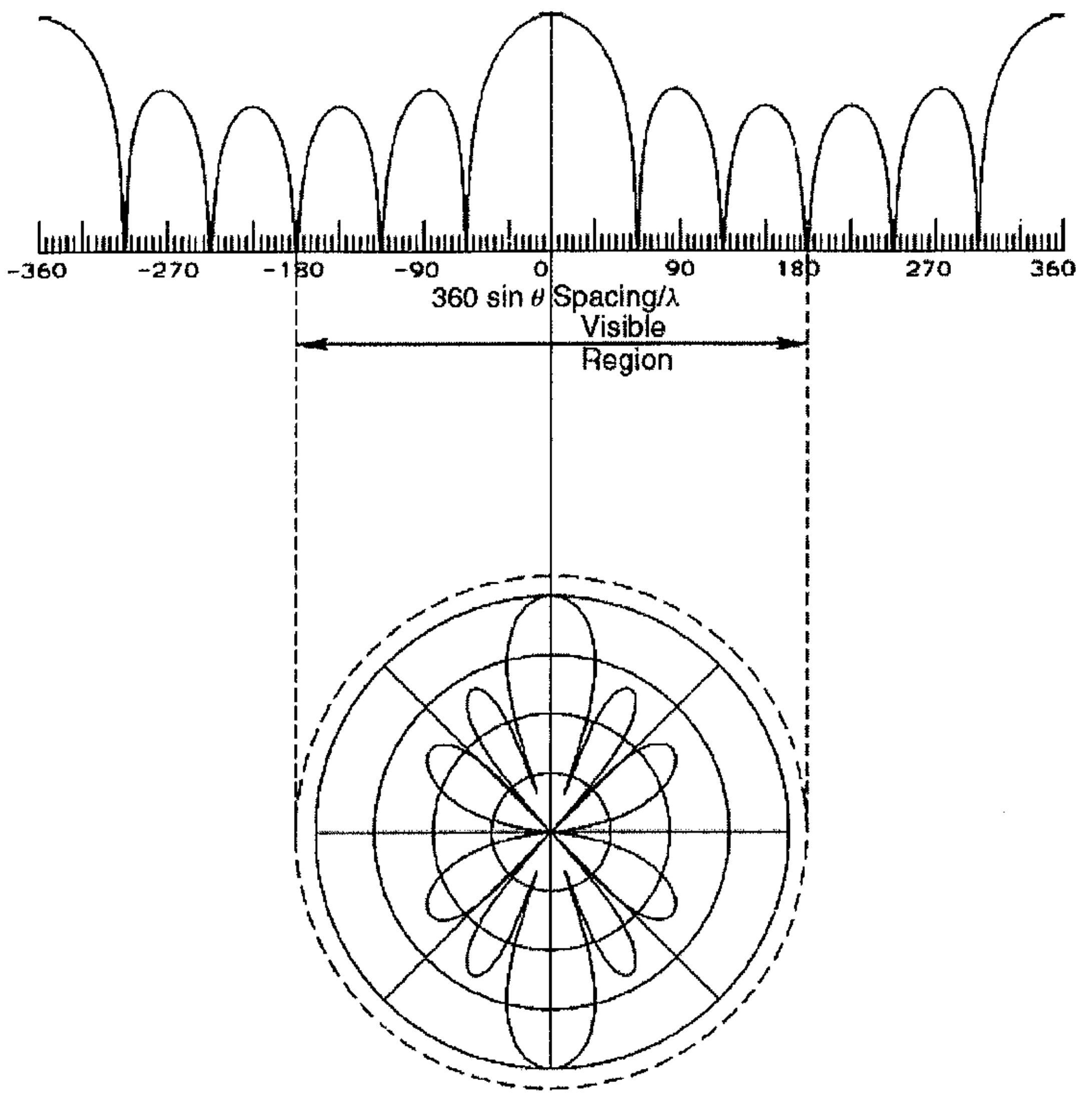


Figure 3 : Circle diagram of a six-element uniform-amplitude array with λ/2 spacing

1.3.2 The Tchebyscheff Distribution

Taylor developed an aperture distribution based on Dolph's use of the Chebyshev polynomials to produce the narrowest beamwidth for a specified sidelobe level for an array. The Chebyshev array design produces equal-amplitude sidelobes that we discover to be undesirable for large arrays because the equivalent aperture distribution peaks at the ends and the average value of the sidelobes limits the directivity to 3 dB above the sidelobe level. Large edge peaking of the distribution requires a feed network containing a large ratio of coupling values. Mutual coupling between elements causes unwanted excitation for a large ratio of element amplitudes and we lose control. Our usual practice is to sample a Taylor distribution for large arrays. The distribution has limited edge peaking, and large arrays can realize high gains. Aperture distribution synthesis involves manipulating pattern nulls to achieve desired characteristics. Taylor used the zeros of the Chebyshev array to alter the positions of the inner nulls of the uniform distribution to lower sidelobe levels.

1.3.3 Tchebyscheff polynomials

Using Euler's formula $\left(e^{ju}\right)^m = e^{j(mu)} = cos(mu) + j\ sin(mu)$

and the trigonometric identity $sin^2 u = 1 - cos^2 u$ *we have*

$$\begin{aligned}
\cos(0u) &= 1 \\
\cos(1u) &= \cos u \\
\cos(2u) &= 2\cos^2 u - 1 \\
\cos(3u) &= 4\cos^3 u - 3\cos u \\
\cos(4u) &= 8\cos^4 u - 8\cos^2 u + 1 \\
\cos(5u) &= 16\cos^5 u - 20\cos^3 u + 5\cos u
\end{aligned}$$

Substituting $z = \cos u$, $\left(u = \frac{kd}{2}\cos \varnothing = \arccos(z)\right)$

we identify the Tschebyscheff polynomials:

$$\cos(mu) = T_m(z)$$

$$\cos(0u) = 1 = T_0(z)$$
$$\cos(1u) = z = T_1(z)$$
$$\cos(2u) = 2z^2 - 1 = T_2(z)$$
$$\cos(3u) = 4z^3 - 3z = T_3(z)$$
$$\cos(4u) = 8z^4 - 8z^2 + 1 = T_4(z)$$
$$\cos(5u) = 16z^5 - 20z^3 + 5z = T_5(z)$$

$$\cos(9u) = 256z^9 - 576z^7 + 432z^5 - 120z^3 + 9z = T_9(z)$$

Thus

$$T_m(x) = \cos(m\cos^{-1}x) \qquad 0 < |x| < 1$$
$$T_m(x) = \cos(m\cos^{-1}x) \qquad 1 < |x| \qquad (1.3.5)$$

Dolph-Tschebyschev Array Design

In this section linear in-phase arrays with nonuniform amplitude distributions are analysed, and the development and application of Dolph-Tchebyscheff distribution are discussed. Let us consider a linear array of an even number n, of isotropic point sources of uniform spacing *d*. All sources are in same phase. The direction θ=0 is taken normal to the array with the origin at the center of the array. The individual sources have the amplitudes A_0, A_1, A_2, A_3... etc.

Even number of elements:

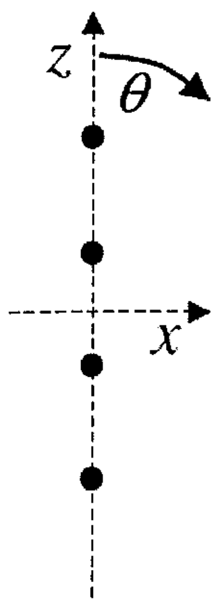


Figure 4 : Even number of elements

Defining $u=\frac{kd}{2}\cos\theta$, $\cos(u)=z$, $\cos(mu)=T_m(z)$

where $T_m(z)$ is the Tschebyscheff polynomial, we can write the array factor as:

$$AF_{2M} = a_1 e^{j\frac{1}{2}kd\cos\theta} + a_2 e^{j\frac{3}{2}kd\cos\theta} + \ldots + a_M e^{j\frac{(2M-1)}{2}kd\cos\theta}$$
$$+ a_1 e^{-j\frac{1}{2}kd\cos\theta} + a_2 e^{-j\frac{3}{2}kd\cos\theta} + \ldots + a_M e^{-j\frac{(2M-1)}{2}kd\cos\theta} \qquad (1.3.6)$$

A common factor of 2 can be normalized and we get:

$$\begin{aligned} AF_{2M} &= \sum_{n=1}^{M} a_n \cos\left[\tfrac{2n-1}{2}kd\cos\theta\right] = \sum_{n=1}^{M} a_n \cos\left[(2n-1)u\right] \\ &= a_1T_1(z) + a_2T_3(z) + a_3T_5(z) + \ldots + a_M T_{(2M-1)}(z) \\ &= \sum_{n=1}^{M} a_n T_{(2n-1)}(z) \end{aligned} \qquad (1.3.7)$$

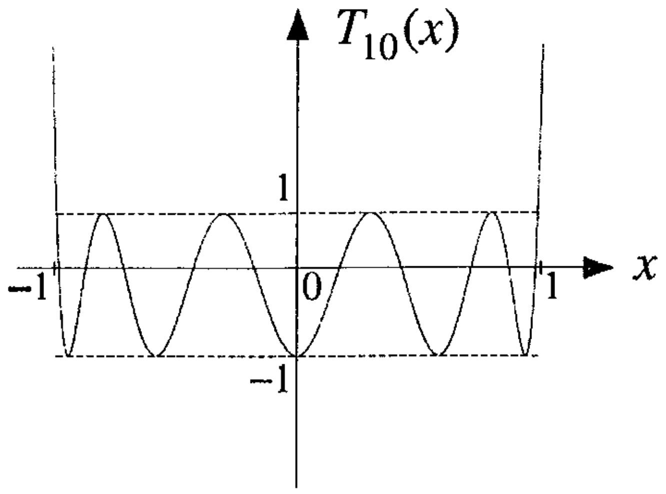


Figure 5 : Chebyshev polynomials of order ten

Only odd order polynomials present.

- Highest order one less than number of elements, 2M.
- Odd number of elements:

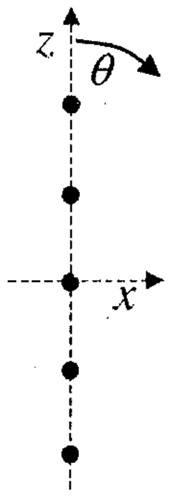


Figure 6 : Odd number of elements

$$AF_{2M+1} = 2a_1 + a_2 e^{jkd\cos\theta} + a_3 e^{j2kd\cos\theta} + \ldots + a_{M+1} e^{jM\,kd\cos\theta} + a_2 e^{-jkd\cos\theta} + a_3 e^{-j2kd\cos\theta} + \ldots + a_{M+1} e^{-jM\,kd\cos\theta} \tag{1.3.8}$$

• Note that the center element has amplitude $2a_1$

• A common factor of 2 can be normalized and we get:

$$\begin{aligned} AF_{2M+1} &= \sum_{n=1}^{M+1} a_n \cos[(n-1)kd\cos\theta] = \sum_{n=1}^{M+1} a_n \cos[2(n-1)u] \\ &= a_1T_0(z) + a_2T_2(z) + a_3T_4(z) + \ldots + a_{M+1}T_{2M}(z) \\ &= \sum_{n=1}^{M+1} a_n T_{2(n-1)}(z) \end{aligned} \tag{1.3.9}$$

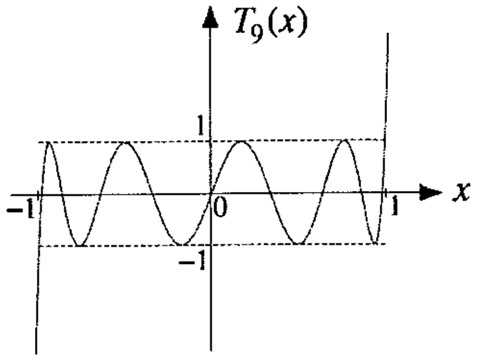


Figure 7 : Chebyshev polynomials of order nine

Only even order polynomials present:

Highest order one less than number of elements, 2M+1. Take this sum of polynomials to be equal to the highest order polynomial present which then will represent the total array factor.

This is possible because only even or odd order polynomials are present.

Procedure for finding the coefficients:

1. Select the appropriate AF according to the number of elements (even or odd).

2. Expand the AF. Replace each *cos(mu)* by its series expansion.

3. Determine the point $z = z_0$ such that $T_N(z_0) = R_0$ (voltage ratio). N is always one less than number of elements.

4. Substitute $cos(u) = z/z_0$ in the expanded AF of step 2.

5. Take the expanded AF to be equal to the Tschebyscheff polynomial of order one less than the number of elements [$T_m(z)$],

in order to find the coefficients.

6. Write down the AF using the found coefficients.

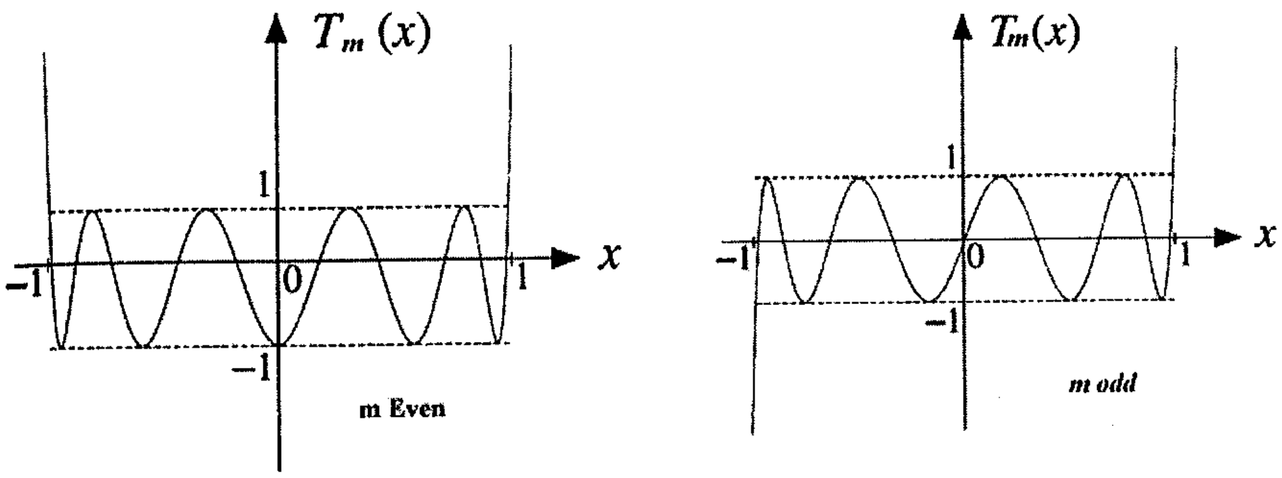


Figure 8 : Tchebyscheff polynomials ,$T_m(x)$ for m even and m odd

Consider the Tchebysceff polynomial of m^th degree

$T_m(x) = cos(m\, cos^{-1} x) = cos(m\delta)$

Where $cos\ \delta = x$

The nulls of the pattern are given by the roots $cos(m\delta) = 0$

That is by $\delta_k^0 = \frac{(2k-1)\pi}{2m}$ k=1,2,........m

The Tchebyscheff polynomials are defined by

$T_m(x) = cos(mcos^{-1}x)$ $0 < |x| < 1$

$T_m(x) = cos(mcos^{-1}x)$ $\qquad 1 < |x|$

By inspection $T_0(x) = 1,\ T_1(x) = x$

The higher–order polynomials can be derived as follows

$T_2(x) = cos(\ 2\ cos^{-1\ x)} = cos\ 2\delta$

$\delta = cos^{\ -1}\ x$ or $x = cos\ \delta$

Now since $cos\ 2\delta = 2cos^2\ \delta\ -1$

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

Similarly, it can be shown that

$T_{m+1}(x) = 2T_m(x)T_1(x) - T_{m-1}(x)$

So that $T_3 = 4x^3 - 3x$ and $T_4(x) = 8x^4 - 8x^2 - 1$ and so on

Now consider the function ψ, For a broadside array

$\psi = \beta d\ cos\ \theta$

As θ varies from 0 to $\pi/2$ to π, ψ goes from βd to 0 to – βd and the range of ψ is $2\beta d$. Now let $x = x_0\ cos\ \psi/2$. Then as θ varies from 0 through $\pi/2$ to π, ψ varies from βd through 0 to $-\beta d$, and x will vary from x_0 cos $\pi d/\lambda$ to x_0 back to $x_0\ cos\ (-\pi d/\lambda) = x_0\ cos\ \pi d/\lambda$.For example, if $d=\lambda/2$, ψ will range from π through zero to $-\pi$ and x will range from 0 to $+x_0$ and back to 0. Again if $d=\lambda$, ψ will range twice around the circle from 2π through 0 to -2π,(two major lobes) and x will range from $-x_0$ to x_0 and back to x_0.

The nulls of the Tchebyshev pattern occur at values of x given by

$x_k^0 = \cos\ \delta_k^0$

So the corresponding position for the nulls on the unit circle will be given by

$x_k^0 = x_0\ \cos\frac{\psi_k^0}{2}$

or $\psi_k^0 = 2\cos^{-1}\frac{x_k^0}{x_0}$

$$\psi_k^0 = 2\cos^{-1}\left[\frac{\cos\delta_k^0}{x_0}\right] \qquad (1.3.10)$$

where $\delta_k^0 = \frac{(2k-1)\pi}{2m}$ k =1,2,3,..........m *(1.3.11)*

The above equation gives the required spacing of nulls on the unit circle for a pattern whose side lobes are all equal. The degree m of the polynomial used will be equal to the number of nulls on the unit circle, and this will be one less than n, the apparent number of elements. The value of x_0 is determined by the desired ratio of b of principal to side lobe amplitudes. The value of x_0 is given in terms of b by

$T_m(x_0) = b$

It can be calculated by noting that if $b = \cosh\rho$, then

$x_0 = \cosh(\rho/m)$

The graphical-analytical method for obtaining the pattern of the from the location of the nulls yields a detailed and accurate plot of relative field strengths versus the defined angle ψ. For many purposes a rough sketch of the pattern may be adequate, and this can be obtained directly from the properties of the Tchebyshev polynomial. Thus, knowing the location of the nulls in the pattern and the amplitude of all the side lobes relative to the principal lobe, the pattern as a function of ψ may be sketched in with good accuracy.

1.4 FILTER SELECTIVITY

A filter's primary purpose is to differentiate between different bands of frequencies, and therefore frequency selectivity is the most common method of classifying filters. Names such as lowpass, highpass, bandpass, and bandstop are used to categorize filters, but it takes more than a name to completely describe a filter. In most cases a precise set of specifications is required in order to allow the proper design of a filter. There are two primary sets of specifications necessary to completely define a filter's response, and each of these can be provided in different ways. The frequency specifications used to describe the passband(s) and stopband(s) could be provided in hertz (Hz) or in radians/second (rad/sec).

The other major filter specifications are the gain characteristics of the passband(s) and stopband(s) of the filter response. A filter's gain is simply the ratio of the output signal level to the input signal level. If the filter's gain is greater than 1, then the output signal is larger than the input signal, while if the gain is less than 1, the output is smaller than the input. In most filter applications, the gain response in the stopband is very small. For this reason, the gain is typically converted to decibels (dB).

$$gain_{dB} = 20 \log(gain) \tag{1.4.1}$$

1.4.1 Lowpass Filters

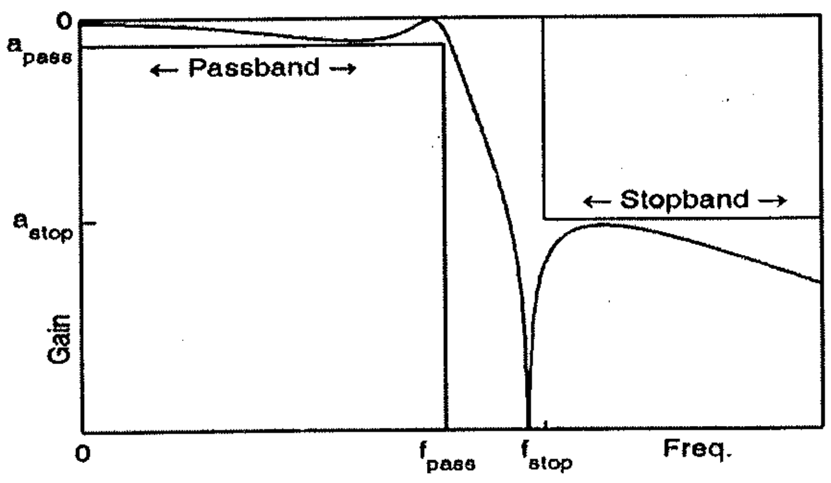


Figure 9 : Low Pass Filter

Figure shows a typical lowpass filter's response using frequency and gain specifications necessary for precision filter design. The frequency range of the filter specification has been divided into three areas. The passband extends from zero frequency (dc) to the passband edge frequency *f*pass, and the stopband extends from the stopband edge frequency *f*stop to infinity. These two bands are separated by the transition band that extends from *f*pass to *f*stop. The filter response within the passband is allowed to vary between 0 dB and the passband gain *a*pass, while the gain in the stopband can vary between the stopband gain *a*stop and negative infinity. (The 0 dB gain in the passband relates to a gain of 1.0, while the gain of negative infinity in the stopband relates to a gain of 0.0.) A lowpass filter's selectivity can now be specified with only four parameters: the passband gain *a*pass, the stopband gain *a*stop, the passband edge frequency *f*pass, and the stopband edge frequency *f*stop.

Applications of Low Pass Filter

Lowpass filters are used whenever it is important to limit the high-frequency content of a signal. For example, if an old audiotape has a lot of high-frequency "hiss," a lowpass filter with a passband edge frequency of 8 kHz could be used to eliminate much of the hiss.

Thus this filter can be used to remove high frequency noise from any speech signal.

1.4.2 Chebyshev Filters

Chebyshev filters are used to separate one band of frequencies from another. Although they cannot match the performance of the windowed-sinc filter, they are more than adequate for many

applications. The primary attribute of Chebyshev filters is their speed, typically more than an order of magnitude faster than the windowed-sinc. This is because they are carried out by recursion rather than convolution. The design of these filters is based on a mathematical technique called the z-transform. The Chebyshev response is a mathematical strategy for achieving a faster rolloff by allowing ripple in the frequency response. Analog and digital filters that use this approach are called Chebyshev filters.

Chebyshev filters have the property that they minimise the error between the idealized filter characteristic and the actual over the range of the filter, but with ripples in the passband. These types of filters are derived from Chebyshev polynomials.

2 DESIGN OF APPLICATION

2.1 FIR FILTER DESIGN

Taking the case of a linear equispaced antenna array with n elements with different amplitude, labelled from left to right as in equation (1.3.3)

$|E| = |A_0 e^{j0} + A_1 e^{j\psi} + A_2 e^{j2\psi} + \ldots\ldots\ldots + A_{n-2} e^{j(n-2)\psi} + A_{n-1} e^{j(n-1)\psi}|$ (2.1.1)

From equation (1.3.2)

$\psi = \frac{2\pi d}{\lambda} \cos\theta + \delta = \beta d \cos\theta + \gamma$ (2.1.2)

where |E| is the magnitude of the far field

$\beta = 2\pi / \lambda$,

λ is the free space wavelength,

d is the spacing between elements,

θ is the angle from the normal to the linear array,

γ is the progressive phase shift from left to right,

and A_0 ,A_1,A_2,..... are complex amplitudes which are proportional to the current amplitudes.

Substituting $z = e^{j\psi}$ then equation (4.1.1) becomes

$H(z) = A_0 + A_1 z + A_2 z^2 + \ldots\ldots + A_{n-2} z^{n-2} + A_{n-1} z^{n-1}$ (2.1.3)

This equation represents the equation (1.2.4)

$H(z^{-1}) = b_0 + b(1)z^{-1} + b(2)z^{-2} + \ldots + b(M)z^{-1}$ which is the equation of a FIR Filter where H(z) is impulse response of the filter with $z=e^{j\omega}$,A_0,A_1,A_2,...........represents amplitudes at the corresponding frequencies.

The Chebyshev Polynomial is given by equation (1.3.5)

$T_m(x) = \cos(m\cos^{-1}x)$ $\quad 0 < |x| < 1$

$T_m(x) = \cos(m\cos^{-1}x)$ $\quad 1 < |x|$ (2.1.4)

The nulls of the Tchebyshev pattern occur at values of x given by equation

(1.3.10) which is

$\psi_k^0 = 2\cos^{-1}\left[\frac{\cos\delta_k^0}{x_0}\right]$

where $\delta_k^0 = \frac{(2k-1)\pi}{2m}$ k =1,2,3,..........m

For the FIR filter $\psi_k^0 = \omega_m$ and $\delta_k^0 = \omega_k$ and m is the order of the filter

Thus

$$\omega_m = 2\cos^{-1}\{\cos(\tfrac{(2k-1)\pi}{2m})/x_0\} \qquad (2.1.5)$$

$\delta_k^0 = \omega_k = (2k-1)\,\pi/2m,$

Thus

$$\omega_m = 2\cos^{-1}\{\cos(\omega_k)/x_0\} \qquad (2.1.6)$$

It can be calculated by noting that if $b = \cosh\rho$, then

$x_0 = \cosh(\rho/m)$.

Substituting x_0 in equation (2.1.6) we have

$$\omega_m = 2\cos^{-1}\{\cos(\omega_k)/\cosh(\rho/m)\} \qquad (2.1.7)$$

and $\rho = \cosh^{-1}b$

substituting ρ in equation (2.1.7) the location of zeros, ω_m, on unit circle can be calculated by the following equation

$$\omega_m = 2\cos^{-1}\{\cos(\omega_k)/\cosh(\cosh^{-1}b\,/m)\} \qquad (2.1.8)$$

Similarly

$$\omega_s = 2\cos^{-1}[1/\{\cosh(1/m\cosh^{-1}b)\}] \qquad (2.1.9)$$

$$\omega_p = 2\cos^{-1}\left[\frac{\cosh\{(1/m)\cosh-1(b/\sqrt{2})\}}{\cosh(1/m\cosh-1b)}\right] \qquad (2.1.10)$$

where

m is the order of the filter

b is the absolute value of attenuation in the stop band,

ω_s is the stopband frequency,

ω_p is the passband frequency.

Using the relation $z_m = e^{j\omega_m}$,we can write equation (2.1.3) as

$$H(z) = (z-z_1)\ (z-z_2) \ldots\ldots (z-z_m) \tag{2.1.11}$$

where,

$z_1, z_2, \ldots$ are location of zeros,

H(z) is the frequency response in z-transform domain.

Replacing z by $e^{j\omega}$ and z_m's by $e^{j\omega_m}$'s in equation (2.1.11)

$$H(z) = (e^{j\omega} - e^{j\omega_1})(e^{j\omega} - e^{j\omega_1}) \ldots (e^{j\omega} - e^{j\omega_m}) \tag{2.1.12}$$

2.1.2 FIR Filter Response With Different Order

keeping b=100 and varying order of filter

Order of the filter m=4

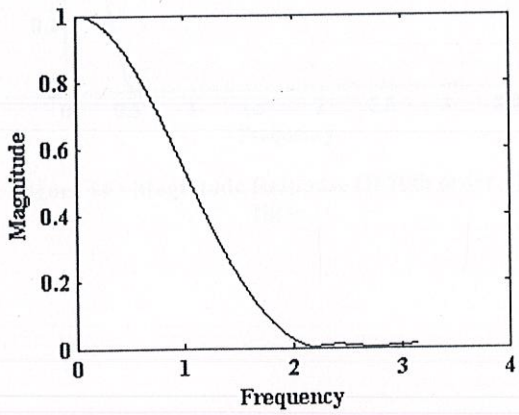


Figure 10 : Magnitude Response Of 4th order FIR filter

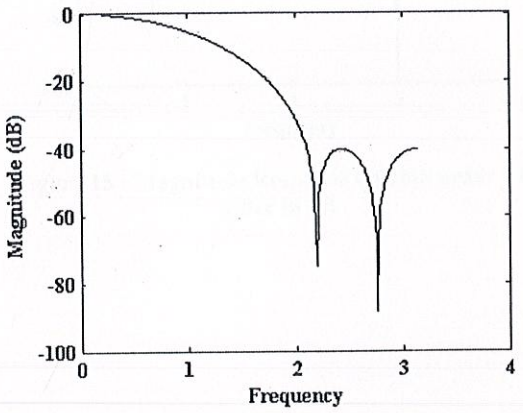


Figure 11 :Magnitude Response Of 4th order FIR filter in dB

Order of the filter m=6

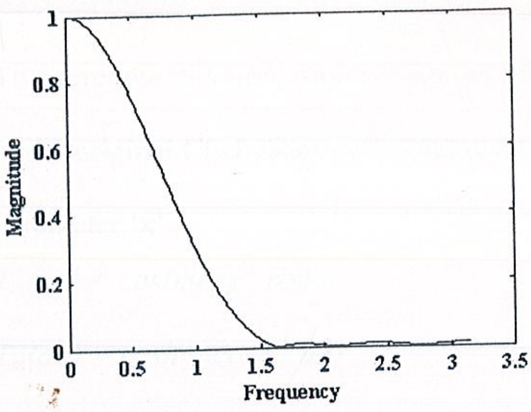


Figure 12 : Magnitude Response Of 6th order FIR filter

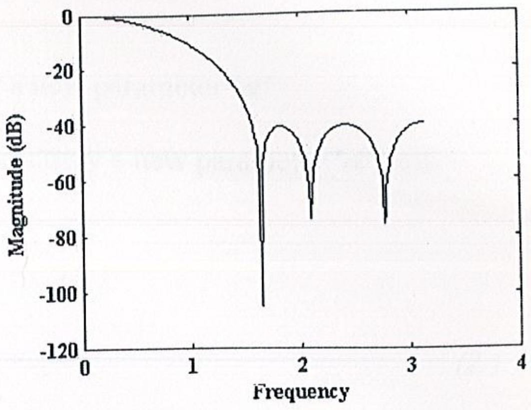


Figure 13 : Magnitude Response Of 6th order FIR filter in dB

Order of the filter m=20

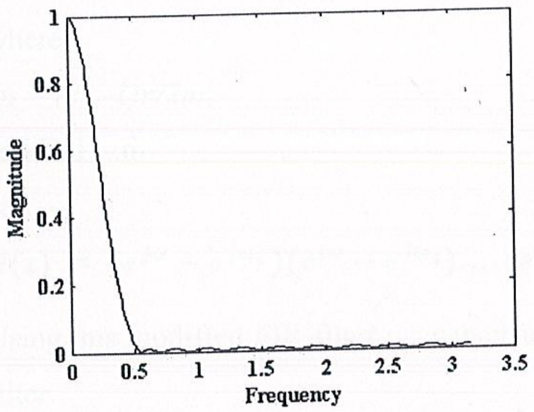


Figure 14 : Magnitude Response Of 20th order FIR filter

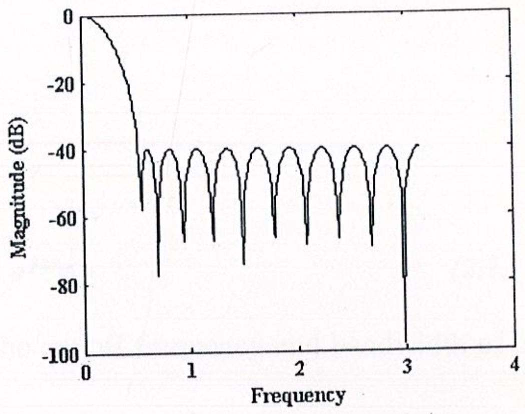


Figure 15 : Magnitude Response Of 20th order FIR filter in dB

2.1.3 Limitation

For a given order filter we cannot change the filter bandwidth. So we cannot change the cutoff frequency for a given order of filter.

2.1.4 Modified Chebyshev FIR Filter

To overcome this limitation we are introducing a new parameter 'α'

In the original Chebyshev polynomial we will multiply a new parameter 'α' with parameter 'x' .

$$T_m(\alpha x) = \cos(m\cos^{-1} \alpha x) \qquad 0 < |x| < 1$$

$$T_m(\alpha x) = \cosh(m\cosh^{-1} \alpha x) \qquad 1 < |x| \qquad (2.1.13)$$

Then ω_s ,ω_m and ω_p becomes

$$\omega_s = 2\cos^{-1}[1/\alpha\{\cosh(1/m\cosh^{-1} b)\}] \qquad (2.1.14)$$

$$\omega_m = 2\cos^{-1}[\cos(\omega_k)/\{ \alpha (\cosh(1/m\cosh^{-1} b))\}] \qquad (2.1.15)$$

$$\omega_p = 2\cos^{-1}\left[\frac{\cosh\{(1/m)\cosh-1(b/\surd 2)\}}{\cosh(1/m\cosh-1b)}\right] \qquad (2.1.16)$$

where

$\omega_k = (2k - 1)\pi/2m$,

and k=1....m.

and

$$H(z) = (e^{j\omega} - e^{j\omega_1})(e^{j\omega} - e^{j\omega_2})....(e^{j\omega} - e^{j\omega_m}) \qquad (2.1.17)$$

Using this modified FIR filter we can change the cuttoff frequency and bandwidth of the filter.

Modified FIR Filter Response Varying Alpha

Order of the filter m=4

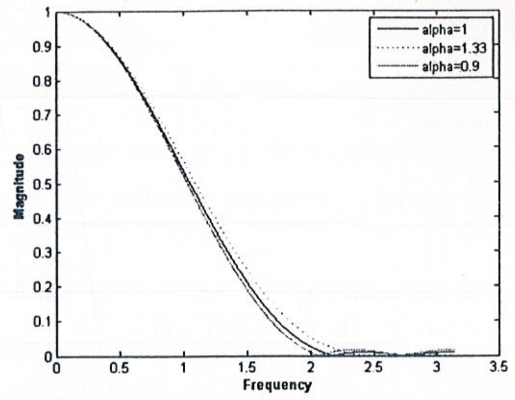


Figure 16 : Magnitude Response Of 4th order FIR filter

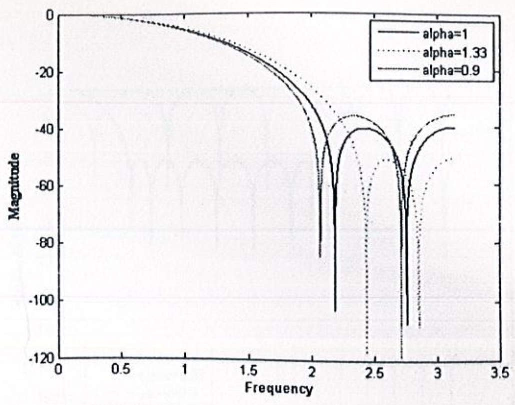


Figure 17 : Magnitude Response Of 4th order FIR filter in dB

We can see that the position of zeroes change with varying the value of alpha.

Order of the filter m=6

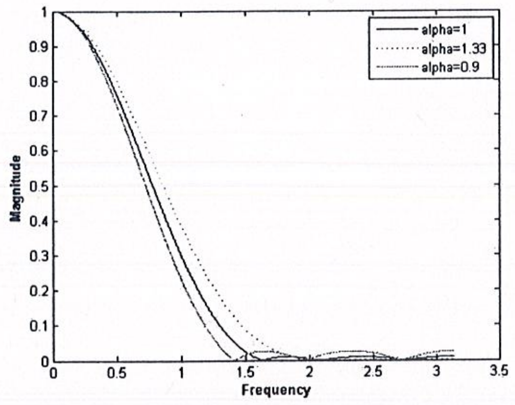


Figure 18 : Magnitude Response Of 6th order FIR filter

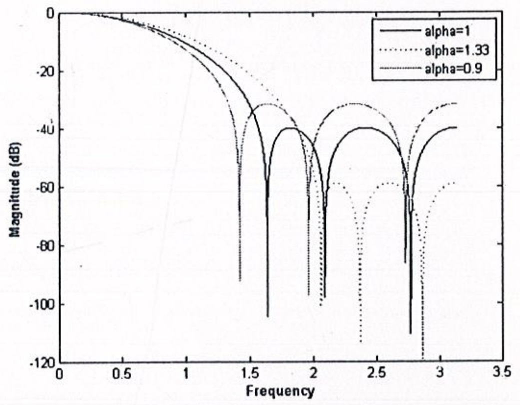


Figure 19 : Magnitude Response Of 6th order FIR filter in dB

Order of the filter m=20

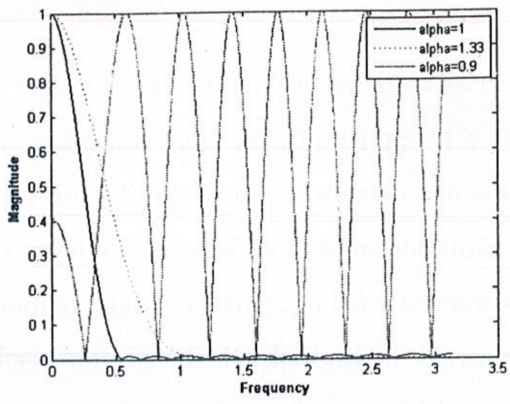


Figure 20 : Magnitude Response Of 20th order FIR filter

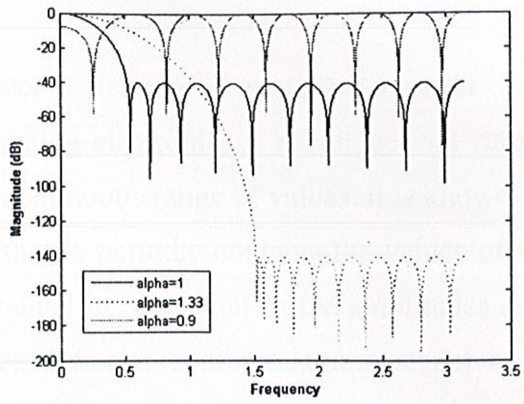


Figure 21 : Magnitude Response Of 20th order FIR filter in dB

2.2 SIGNAL AND NOISE ANALYSIS

2.2.1 Signals

A signal is a description of how one parameter is related to another parameter. For example, the most common type of signal in analog electronics is a voltage that varies with time. Since both parameters can assume a continuous range of values, it is known as a continuous signal. A continuous-time signal that is periodic contains the values of its fundamental frequency and the harmonics contained in it, as well as the amplitudes and phase angles of the individual harmonics. We define a continuous-time signal as a function of an independent variable that is continuous. A one-dimensional continuous-time signal f (t) is expressed as a function of time that varies continuously from $-\infty$ to ∞. But it may be a function of other variables such as temperature, pressure, or elevation. In comparison, passing this signal through an analog-to-digital converter forces each of the two parameters to be quantized.

For instance, imagine the conversion being done with 12 bits at a sampling rate of 1000 samples per second. The voltage is curtailed to 4096 (212) possible binary levels and the time is only defined at one millisecond increments. Signals formed from parameters that are quantized in this manner are said to be discrete signals or digitized signals. For the most part, continuous signals exist in nature, while discrete signals exist inside computers (although you can find exceptions to both cases). It is also possible to have signals where one parameter is continuous and the other is discrete.

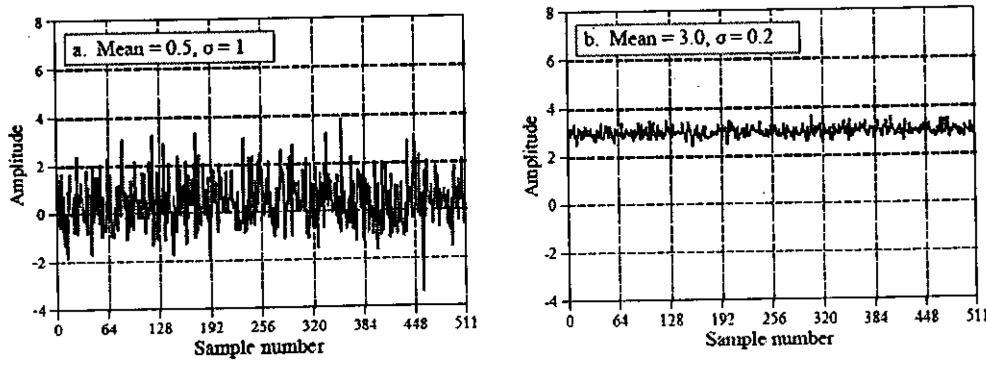


Figure 22 : Shows two discrete signals, such as might be acquired with a digital data acquisition system

The vertical axis may represent voltage, light intensity, sound pressure, or an infinite number of other parameters. Since we don't know what it represents in this particular case, we will give it the generic label: amplitude.

This parameter is also called several other names: the y-axis, the dependent variable, the range, and the ordinate. The horizontal axis represents the other parameter of the signal, going by such names as: the x-axis, the independent variable, the domain, and the abscissa. Time is the most common parameter to appear on the horizontal axis of acquired signals; however, other parameters are used in specific applications.

Time Domain/Frequency domain

A signal that uses time as the independent variable (i.e., the parameter on the horizontal axis), is said to be in the time domain. Another common signal uses frequency as the independent variable, resulting in the term, frequency domain. Likewise, signals that use distance as the independent parameter are said to be in the spatial domain (distance is a measure of space).

The type of parameter on the horizontal axis *is* the domain of the signal. Sometimes the x-axis is labeled with something very generic, such as sample number? Such signals can be referred to as being in the time domain. This is because sampling at equal intervals of time is the most common way of obtaining signals. Each sample is assigned a sample number or index. The variable, N, is widely used to represent the total number of samples in a signal.

Sampling theorem

Sampling Rate

The sampling rate, sample rate, or sampling frequency defines the number of samples per second (or per other unit) taken from a continuous signal to make a discrete signal. For time-domain signals, it can be measured in hertz (Hz). The inverse of the sampling frequency is the sampling period or sampling interval, which is the time between samples.

The Sampling Frequency for a given signal (continuous) should be in accordance with the Nyquist criterion, following which, will ensure adequate samples in the discrete domain such that the signal can be perfectly reconstructed.

The Nyquist–Shannon sampling theorem states that perfect reconstruction of a signal is possible when the sampling frequency is greater than twice the bandwidth of the signal being sampled, or equivalently, that the Nyquist frequency (half the sample rate) exceeds the bandwidth of the signal being sampled. If lower sampling rates are used, the original signal's information may not be completely recoverable from the sampled signal. For example, if a signal has a bandwidth of 100 Hz, to avoid aliasing the sampling frequency should be greater than 200 Hz.

Analog to digital conversion

Analog-to-digital conversion is an electronic process in which a continuously variable (analog) signal is changed, without altering its essential content, into a multi-level (digital) signal.

The input to an analog-to-digital converter (ADC) consists of a voltage that varies among a theoretically infinite number of values. Examples are sine waves, the waveforms representing human speech, and the signals from a conventional television camera. The output of the ADC, in contrast, has defined levels or states. The number of states is almost always a power of two -- that is, 2, 4, 8, 16, etc. The simplest digital signals have only two states, and are called binary. All whole numbers can be represented in binary form as strings of ones and zeros.

Digital signals propagate more efficiently than analog signals, largely because digital impulses, which are well-defined and orderly, are easier for electronic circuits to distinguish from noise, which is chaotic. This is the chief advantage of digital modes in communications.

Pulse Code Modulation

Modulation

Pulse-code modulation (PCM) is a digital representation of an analog signal where the magnitude of the signal is sampled regularly at uniform intervals, then quantized to a series of symbols in a digital (usually binary) code.

In the following diagram, a sine wave (red curve) is sampled and quantized for PCM. The sine wave is sampled at regular intervals, shown as ticks on the x-axis. For each sample, one of the available values (ticks on the y-axis) is chosen by some algorithm. This produces a fully discrete representation of the input signal (shaded area) that can be easily encoded as digital data for storage or manipulation. For the sine wave example at right, we can verify that the quantized values at the sampling moments are 9, 11, 12, 13, 14, 14, 15, 15, 15, 14, etc. Encoding these values as binary numbers would result in the following set of nibbles: 1001, 1011, 1100, 1101, 1110, 1110, 1111, 1111, 1111, 1110...

There are many ways to implement a real device that performs this task. In real systems, such a device is commonly implemented on a single integrated circuit that lacks only the clock necessary for sampling, and is generally referred to as an ADC (analog-to-digital converter). These devices will produce on their output a binary representation of the input whenever they are triggered by a clock signal, which would then be read by a processor of some sort.

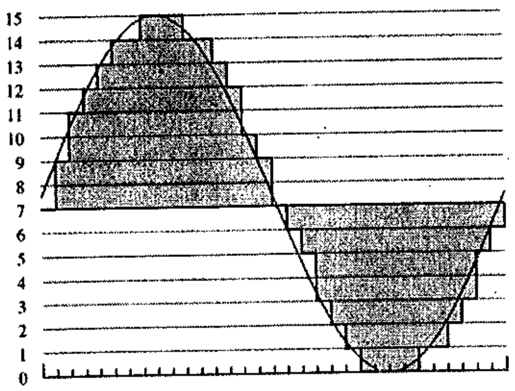


Figure 23 : Sampling and quantization of a signal (red) for 4-bit PCM

Demodulation

To produce output from the sampled data, the procedure of modulation is applied in reverse. After each sampling period has passed, the next value is read and the output of the system is shifted instantaneously (in an idealized system) to the new value.

The electronics involved in producing an accurate analog signal from the discrete data are similar to those used for generating the digital signal. These devices are DACs (digital-to-analog converters), and operate similarly to ADCs. They produce on their output a voltage or current (depending on type) that represents the value presented on their inputs. This output would then generally be filtered and amplified for use.

Quantization

In digital signal processing, quantization is the process of approximating a continuous range of values (or a very large set of possible discrete values) by a relatively-small set of discrete symbols or integer values.

A common use of quantization is in the conversion of a discrete signal (a sampled continuous signal) into a digital signal by quantizing. Both of these steps (sampling and quantizing) are performed in analog-to-digital converters with the quantization level specified in bits. A specific example would be compact disc (CD) audio which is sampled at 44,100 Hz and quantized with 16 bits (2 bytes) which can be one of 65,536 (i.e. 2^{16}) possible values per sample.

Discrete Fourier Transform

The discrete Fourier transform (DFT), occasionally called the finite Fourier transform, is a transform for Fourier analysis of finite-domain discrete-time signals. It is widely employed in signal processing and related fields to analyze the frequencies contained in a sampled signal, to solve partial differential equations, and to perform other operations such as convolutions. The DFT can be computed efficiently in practice using a fast Fourier transform (FFT) algorithm.

Since FFT algorithms are so commonly employed to compute the DFT, the two terms are often used interchangeably in colloquial settings, although there is a clear distinction:

"DFT" refers to a mathematical transformation, regardless of how it is computed, while "FFT" refers to any one of several efficient algorithms for the DFT.

The sequence of N complex numbers $x_0, \ldots, x_{N-1}$ is transformed into the sequence of N complex numbers $X_0, \ldots, X_{N-1}$ by the DFT according to the formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1 \tag{2.2.1}$$

Where e is the base of the natural logarithm, i is the imaginary unit ($i^2 = -1$), and π is pi. The transform is sometimes denoted by the symbol$\mathcal{F}$, as in $\mathbf{X} = \mathcal{F}\{\mathbf{x}\}$or $\mathcal{F}(\mathbf{x})$or $\mathcal{F}\mathbf{x}$.

The *inverse discrete Fourier transform (IDFT)* is given by

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k e^{\frac{2\pi i}{N} kn} \qquad n = 0, \ldots, N-1. \tag{2.2.2}$$

Note that the normalization factor multiplying the DFT and IDFT (here 1 and $1/N$) and the signs of the exponents are merely conventions, and differ in some treatments. The only requirements of these conventions are that the DFT and IDFT have opposite-sign exponents and that the product of their normalization factors be $1/N$. A normalization of $1/\sqrt{N}$ for both the DFT and IDFT makes the transforms unitary, which has some theoretical advantages, but it is often more practical in numerical computation to perform the scaling all at once as above.

Fast Fourier Transform

A Fast Fourier Transform (FFT) is an efficient algorithm to compute the discrete Fourier transform (DFT) and it's inverse. FFT's are of great importance to a wide variety of applications, from digital signal processing to solving partial differential equations to algorithms for quickly multiplying large integers.

Let $x_0, \ldots, x_{N-1}$ be complex numbers. The DFT is defined by the equation

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N}nk} \qquad k = 0, \dots, N-1.$$

Evaluating these sums directly would take O(N^2) arithmetical operations. An FFT is an algorithm to compute the same result in only O(N log N) operations. In general, such algorithms depend upon the factorization of N, but (contrary to popular misconception) there are FFTs with O(N log N) complexity for all N, even for prime N.

Many FFT algorithms only depend on the fact that $e^{-\frac{2\pi i}{N}}$ is a primitive root of unity, and thus can be applied to analogous transforms over any finite field, such as number-theoretic transforms.

Since the inverse DFT is the same as the DFT, but with the opposite sign in the exponent and a 1/N factor, any FFT algorithm can easily be adapted for it as well.

FFT algorithms

The most commonly used FFT algorithms comprise of the following:

Cooley-Tukey FFT algorithm

By far the most common FFT is the Cooley-Tukey algorithm. This is a divide and conquer algorithm that recursively breaks down a DFT of any composite size $N = N_1N_2$ into many smaller DFTs of sizes N_1 and N_2, along with O(N) multiplications by complex roots of unity traditionally called twiddle factors.

The most well-known use of the Cooley-Tukey algorithm is to divide the transform into two pieces of size N / 2 at each step, and is therefore limited to power-of-two sizes, but any factorization can be used in general (as was known to both Gauss and Cooley/Tukey). These are called the radix-2 and mixed-radix cases, respectively (and other variants such as the split-radix FFT have their own names as well). Although the basic idea is recursive, most traditional implementations rearrange the algorithm to avoid explicit recursion. Also, because the Cooley-Tukey algorithm breaks the DFT into smaller DFTs, it can be combined arbitrarily with any other algorithm for the DFT.

Goertzel algorithm

The Goertzel algorithm computes a sequence, $s(n)$, given an input sequence, $x(n)$, as

$$s(n) = x(n) + 2\cos(2\pi\omega)s(n-1) - s(n-2) \tag{2.2.3}$$

Where $s(-2) = s(-1) = 0$ and ω is some frequency of interest, normalized with respect to the sampling frequency. This effectively implements a second-order IIR filter with poles at $e^{+2\pi i\omega}$ and $e^{-2\pi i\omega}$, and requires only one multiply (assuming $2cos(\omega)$ is pre-computed), one addition and one subtraction per input sample. For real inputs, these operations are real.

The Z transform of this process is

$$\frac{S(z)}{X(z)} = \frac{1}{1-2\cos(2\pi\omega)z^{-1}+z^{-2}} = \frac{1}{(1-e^{+2\pi i\omega}z^{-1})(1-e^{-2\pi i\omega}z^{-1})} \tag{2.2.4}$$

Applying an additional, FIR, transform of the form

$$\frac{Y(z)}{S(z)} = 1 - e^{-2\pi i\omega}z^{-1} \tag{2.2.5}$$

Will give an overall transform of

$$\frac{S(z)}{X(z)}\frac{Y(z)}{S(z)} = \frac{Y(z)}{X(z)} = \frac{(1-e^{-2\pi i\omega}z^{-1})}{(1-e^{+2\pi i\omega}z^{-1})(1-e^{-2\pi i\omega}z^{-1})} = \frac{1}{1-e^{+2\pi i\omega}z^{-1}} \tag{2.2.6}$$

The time-domain equivalent of this overall transform is

$$y(n) = x(n) + e^{+2\pi i\omega}y(n-1) = \sum_{k=-\infty}^{n} x(k)e^{+2\pi i\omega(n-k)}$$

$$= e^{+2\pi i\omega n}\sum_{k=-\infty}^{n} x(k)e^{-2\pi i\omega k} \tag{2.2.7}$$

Which, when $x(n) = 0$ for all $n < 0$, becomes

$$y(n) = e^{+2\pi i\omega n}\sum_{k=0}^{n} x(k)e^{-2\pi i\omega k} \tag{2.2.8}$$

Or, the equation for the $(n + 1)$-sample DFT of x, evaluated for ω and multiplied by the scale factor $e^{+2\pi i \omega n}$.

Other FFT algorithms

- Prime-factor FFT algorithm,
- Bruun's FFT algorithm,
- Rader's FFT algorithm,
- Bluestein's FFT algorithm.

2.2.2 Noise Analysis

White noise

White noise is a random signal (or process) with a flat power spectral density. In other words, the signal's power spectral density has equal power in any band, at any centre frequency, having a given bandwidth. White noise is considered analogous to white light which contains all frequencies.

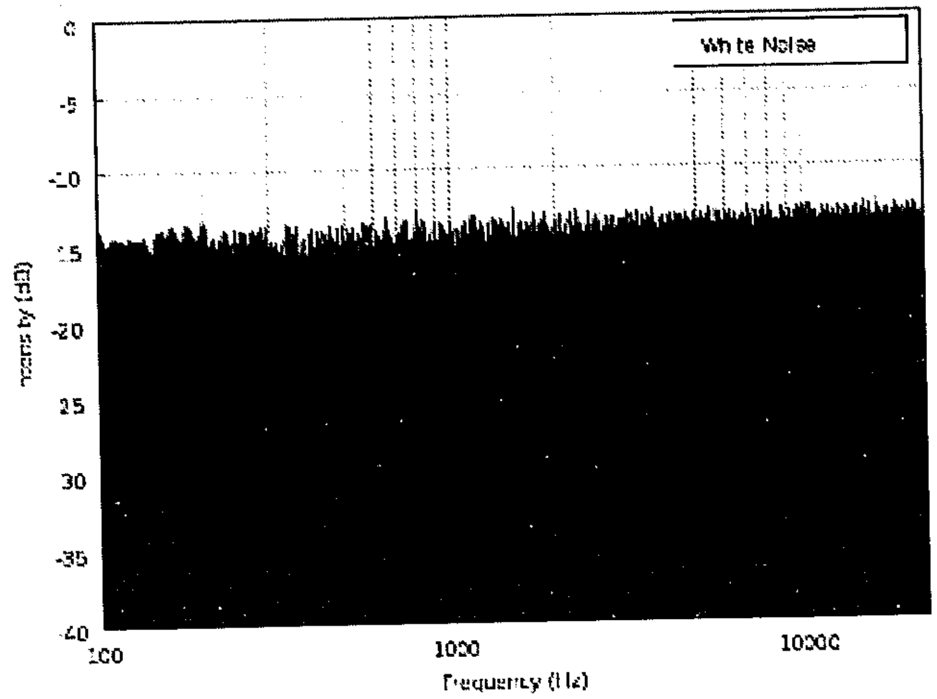


Figure 24 : Calculated spectrum of a generated approximation of white noise

An infinite-bandwidth white noise signal is purely a theoretical construction. By having power at all frequencies, the total power of such a signal is infinite. In practice, a signal can be "white" with a flat spectrum over a defined frequency band.

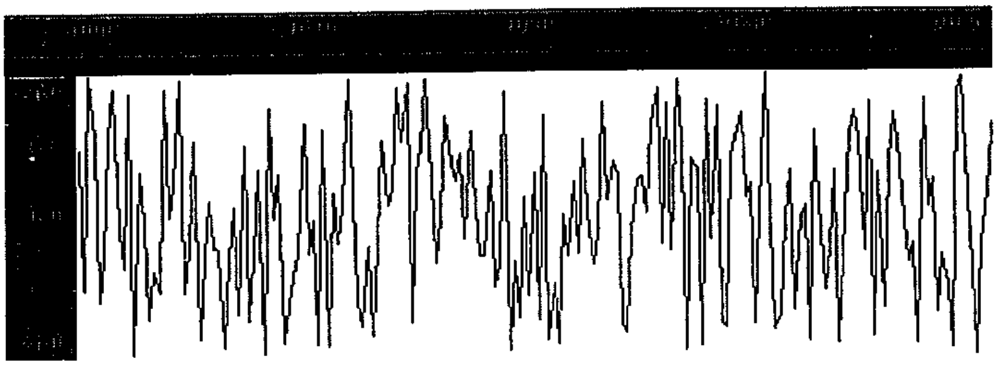


Figure 25 : Four thousandths of a second of white noise

The term white noise is also commonly applied to a noise signal in the spatial domain which has an autocorrelation which can be represented by a delta function over the relevant space dimensions. The signal is then "white" in the spatial frequency domain (this is equally true for signals in the angular frequency domain, e.g. the distribution of a signal across all angles in the night sky). The image below displays a finite length, discrete time realization of a white noise process generated from a computer.

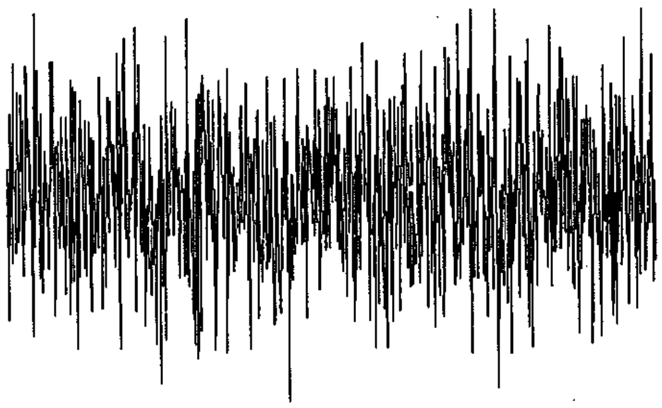


Figure 26 : An example realization of a white noise process

Thermal Noise

Thermal noise is the Electronic noise generated by the thermal agitation of the charge carriers (the electrons) inside an electrical conductor in equilibrium, which happens regardless of any applied voltage.

Thermal noise is approximately white, meaning that the power spectral density is equal throughout the frequency spectrum. Additionally, the amplitude of the signal has very nearly a Gaussian probability density function.

Burst Noise

Burst noise is a type of electronic noise that occurs in semiconductors. It is also called impulse noise, bi-stable noise, or random telegraph signals (RTS noise). It consists of sudden step-like transitions between two or more levels (non-Gaussian), as high as several hundred microvolts, at random and unpredictable times. Each shift in offset voltage or current lasts for several milliseconds.

2.2.3 Input Signal Analysis

The input signal is a speech signal in time domain represented as in the figure below:

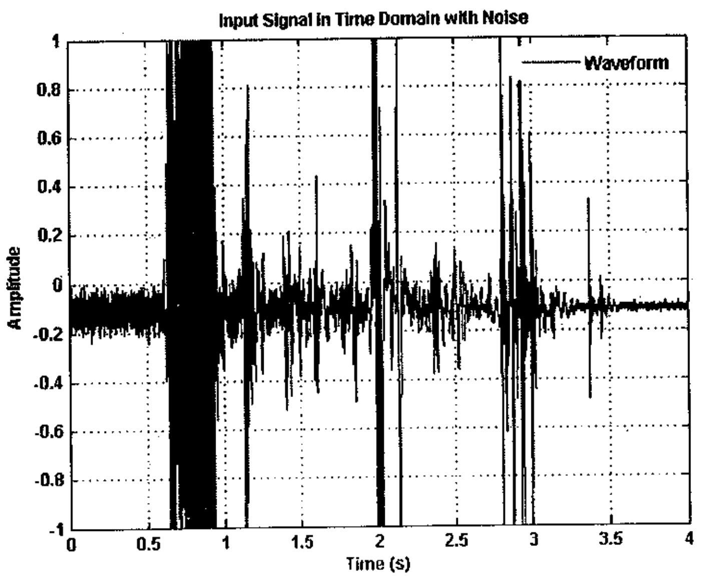


Figure 27 : Speech signal in time domain with only white noise

The graph describes a discrete time signal which has been sampled off a continuous speech (recorded using a traditional microphone) signal at a sampling frequency 8000Hz using Pulse Code Modulation (PCM) at 8 Bits in the mono mode. The duration of the signal is 4seconds, and the total number of samples are equal to 32000 (= 4seconds* 8000Hz). Here, the maximum desirable value of the frequency is 3500Hz to 4000Hz (i.e., towards the lower end of the frequency spectrum or a male human voice) thus, resulting in a sampling frequency of about 8000Hz. Also, the recorded signal has been corrupted by white noise (hissing noise).

On corrupting the above signal with an arbitrary co sinusoidal noise (modeled as Burst Noise) centered at frequency of 3000Hz, the following signal results:

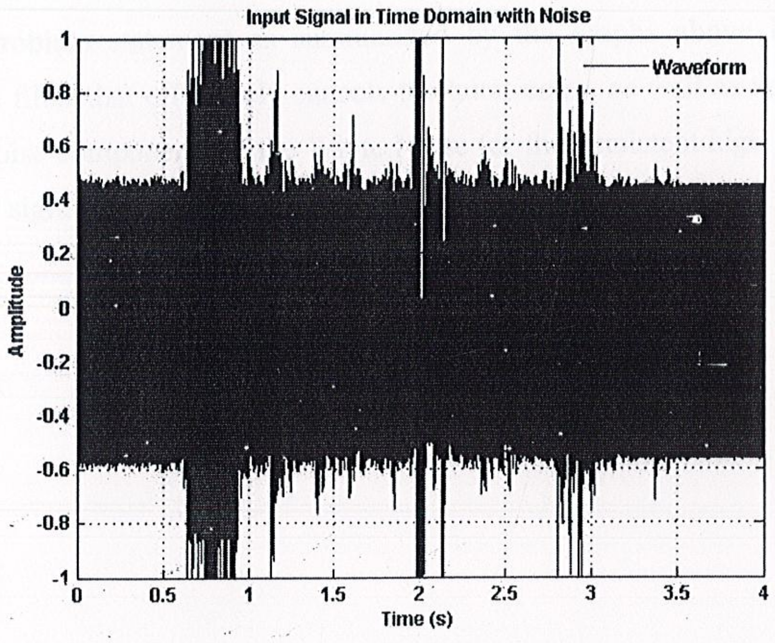


Figure 28 : Speech signal in time domain with both the white noise as well as the burst noise

As is clearly visible, the Signal to Noise ratio decreases, making the original speech almost inaudible. The same can be more lucidly displayed by representing the signal in frequency domain. Thus, applying a 32767 point FFT on the above signal, we get the following:

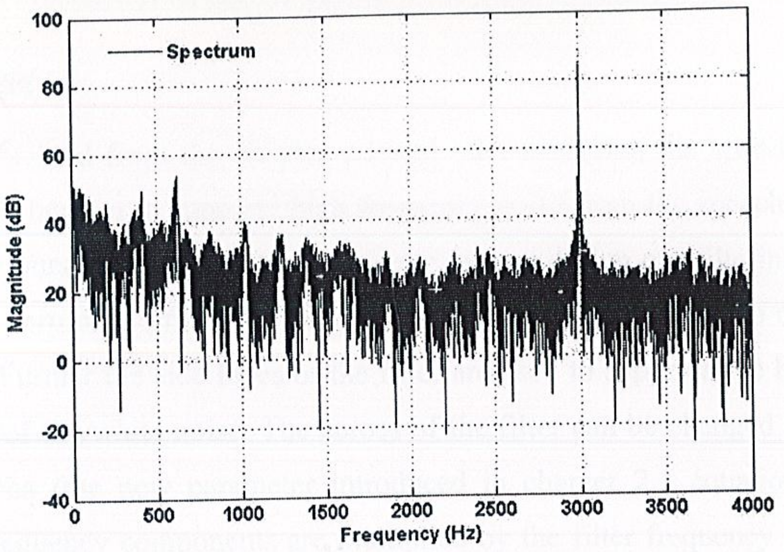


Figure 29 : Speech signal in frequency domain with burst noise centered at 3000Hz

Thus, the problem statement as summarized by the graphs above is to design and implement a filter that effectively cancels the burst noise, as well as attenuates the high frequency noise components of the White Noise (or the persistent high frequency hiss in the recorded signal).

2.3 PASSING SPEECH SIGNAL THROUGH THE FIR FILTER

2.3.1 Concept

The Filter obtained from the chapter 2.1 and after analysing the speech signal we can design the application to suppress high frequency noise from the speech signal. In order to cancel the burst noise at a given frequency we can design our filtering such a way so that the first zero of the filter lies on that particular frequency so as to cancel that noise completely. Further the side lobes of the filter are used to suppress the higher frequency components of the white noise. The zeroes of the filter can be changed by changing the value of alpha (the new parameter introduced in chapter 2.1 equation 2.1.13) .The individual frequency components are multiplied by the filter frequency response so that the lower frequency signal had a 0 dB attenuation while the higher frequency signal has a significant attenuation in the frequency spectrum. To achieve this result, fast Fourier transform of the speech signal is taken and the individual frequency component is multiplied with its corresponding filter response in z domain.

2.3.2 Steps involved

a. Take FFT of the speech signal

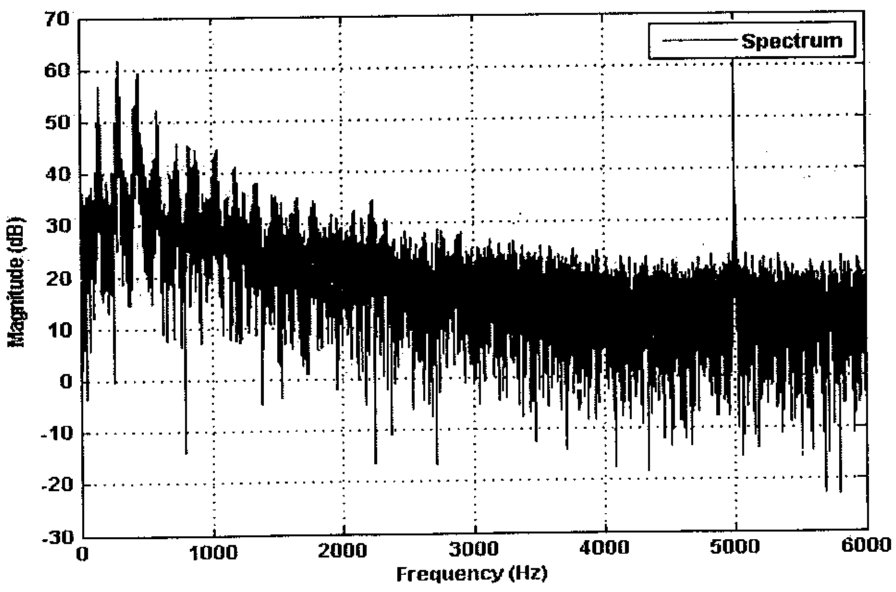


Figure 30 : FFT of the speech signal

The number of samples in the time domain is usually represented by the variable N.

Here N=32000 samples

While N can be any positive integer, a power of two is usually chosen, i.e., 128, 256, 512, 1024, etc. There are two reasons for this.

- Digital data storage uses binary addressing; making powers of two a natural signal length.
- The most efficient algorithm for calculating the DFT, the Fast Fourier Transform (FFT), usually operates with N that is a power of two.

Here we take 32768 point FFT of the audio signal

b. Plotting Filter Response

Order of Filter: 6

Location Of First Zero: 3000Hz (To cancel the sinusoidal noise at 3000Hz)

Alpha: 1.78

Attenuation: 40dB

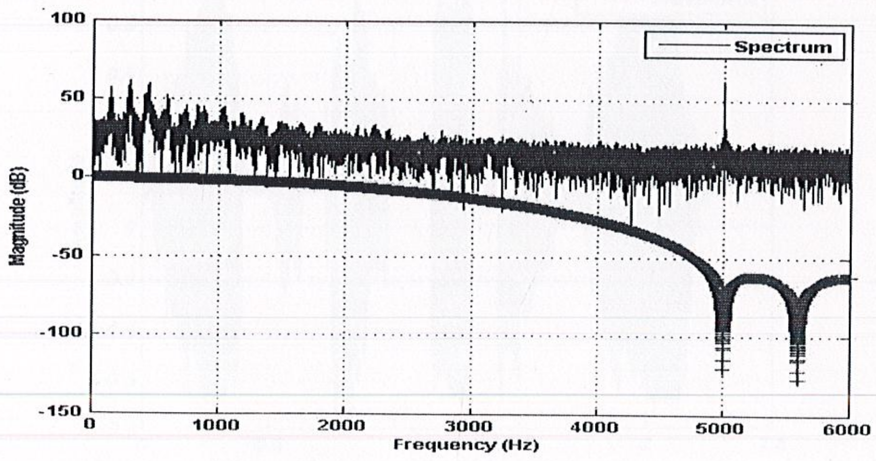


Figure 31 : Filter Response

c. Frequency multiplication

Multiplying individual frequency components of the speech signal with the filter response we get

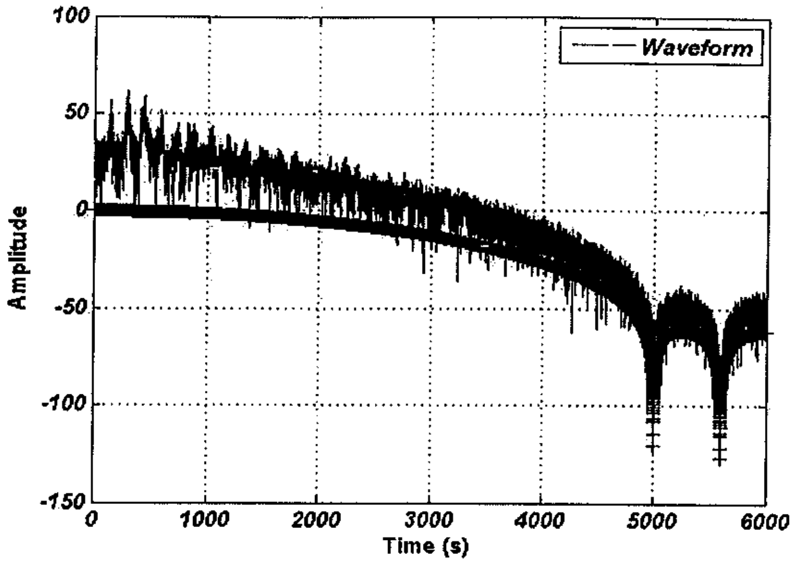


Figure 32 : Frequency multiplication

d. Taking in inverse FFT (IFFT) of the resultant signal

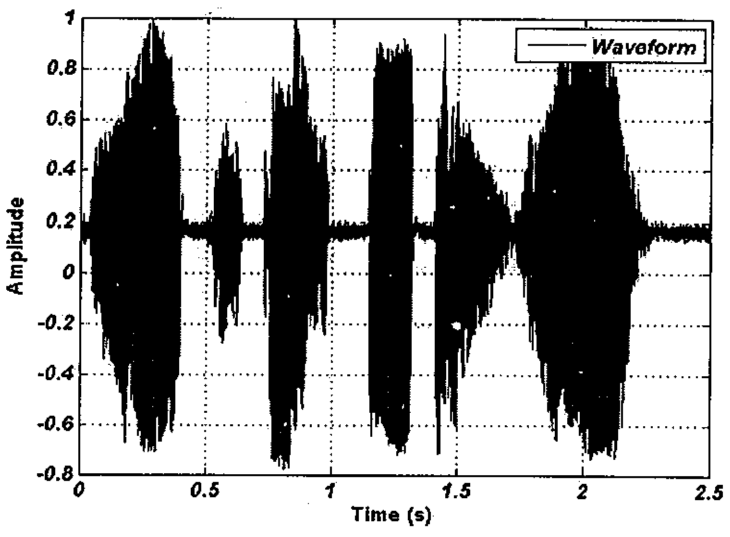


Figure 33 : Inverse FFT (IFFT) of the resultant signal

2.3.3 Time Domain representation of entire process

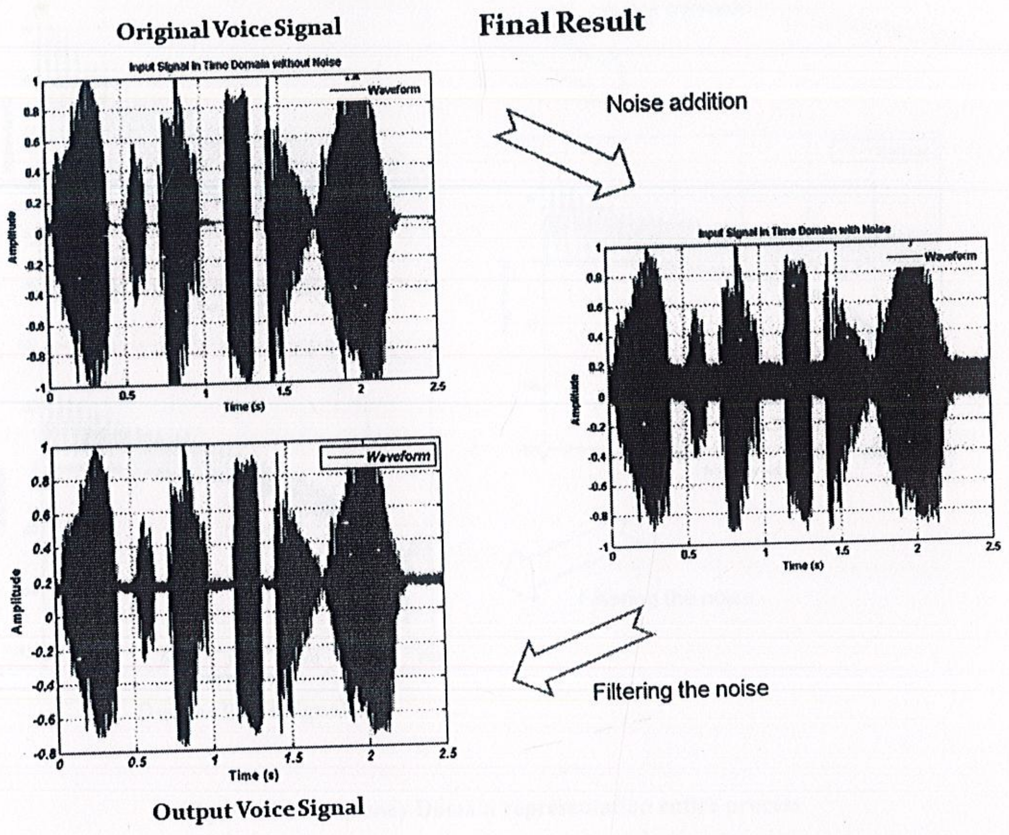


Figure 34 : Time Domain representation of entire process

2.3.4 Frequency Domain representation entire process

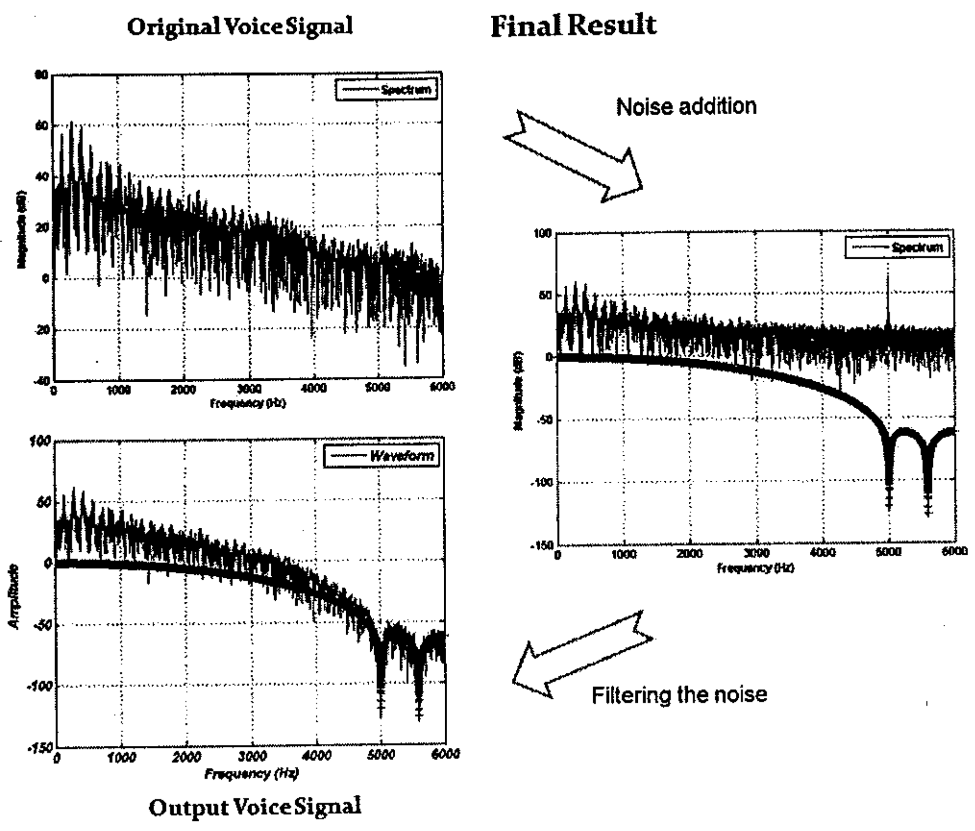


Figure 35 : Frequency Domain representation entire process

3 IMPLEMENTATION

3.1 MAIN APPLICATION SOURCE CODE

```
function varargout = untitled3(varargin)

% Begin initialization code
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @untitled3_OpeningFcn, ...
                   'gui_OutputFcn',  @untitled3_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

% End initialization code
% Executes just before untitled3 is made visible.
% ---------------------------------------------------------------------

function file_menu_Callback(hObject, eventdata, handles)
% hObject    handle to file_menu (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data
set(handles.axes8,'HandleVisibility','OFF');

% get file from user
clear global;
global FileName
global PathName
[FileName,PathName] = uigetfile('*.wav','Select the Wav-file');

% reading wav file
global x
global fs
[x,fs]=wavread(fullfile(PathName, FileName));
global t3
t3=(0:length(x)-1)/fs;

% plotting original signal
set(handles.axes16,'HandleVisibility','ON');
axes(handles.axes16);
axis on;
plot(t3,x/(max(x)),'Color',[0.3137 0.3137 0.3137]);
grid on;
clc;
```

```
function untitled3_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data
% varargin   command line arguments to untitled3
% Choose default command line output for untitled3
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes untitled3 wait for user response (see UIRESUME)
% uiwait(handles.figure1);

function order_Callback(hObject, eventdata, handles)
% hObject    handle to order
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data
% --- Executes during object creation, after setting all properties.

function order_CreateFcn(hObject, eventdata, handles)
% hObject    handle to order (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called
% Hint: edit controls usually have a white background on Windows.

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in ok.
function ok_Callback(hObject, eventdata, handles)
% hObject    handle to ok (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global u
global x
global fs
t=(0:length(x)-1)/fs;
L=length(x);

% fourier transform of signal
NFFT = 2^nextpow2(L);
Y=fft(u,NFFT);

% plotting frequecy spectrum
f = fs/2*linspace(0,1,NFFT/2);
om22 = str2num(get(handles.order,'String'));
be = str2num(get(handles.db,'string'));
fq = str2num(get(handles.cutoff,'string'));
be2=10^(be/20);
Fs2=length(f);
```

```
temp1=(fq*Fs2)/max(f);
temp2=(pi*temp1)/Fs2;
yelpha=(cos(pi/(2*om22)))/((cos(temp2/2))*(cosh((1/om22)*acosh(be2))));
set(handles.alpha,'String',yelpha)
fnew=20*log10(abs(Y(1:length(f))));
set(handles.axes6,'HandleVisibility','ON');
axes(handles.axes6);
plot(f,fnew,'Color',[0.3137 0.3137 0.3137]);
grid on;

%Filter Design Starts Here
b=be2;
m=om22;
a=[1:m];
i=1;
freq=0;
Fs2=length(f);


h=1;
p=1;
omegak=((2*a-1)*pi)/(2*m);
omegam=2*acos((cos(omegak))/(((yelpha)*(cosh(1/m*acosh(b))))));
while(freq<Fs2)
    om=2*pi*freq/(2*Fs2);
    k=exp(j*om);
    while(i<=m)
        h=h*(k-exp(j*omegam(i)));
        i=i+1;
    end
    h1(p)=abs(h);
    p=p+1;
    freq=freq+1;
    h=1;
    i=1;
end
h2=(h1/max(h1));
h3=20*log10(h2);;
set(handles.axes11,'HandleVisibility','ON');
axes(handles.axes11);
axis tight;
plot(f,h3,'Color',[0.3137 0.3137 0.3137]);
grid on;
mul=(Y(1:length(f))).*h2';
mul2=abs(mul);
mulnew=20*log10(mul2);
set(handles.axes13,'HandleVisibility','ON');
axes(handles.axes13);
axis tight;
plot(f,mulnew,'Color',[0.3137 0.3137 0.3137]);
grid on;
mul3=ifft(mul,NFFT);
mul4=abs(mul3);
mul5=max(mul4);
mul6=mul4/mul5;

rmul=real(mul3(1:length(x)));
```

```
set(handles.axes12,'HandleVisibility','ON');
axes(handles.axes12);
axis tight;
plot(t,rmul/(max(rmul)),'Color',[0.3137 0.3137 0.3137]);
grid on;
global rfinal
rfinal=rmul/(max(rmul));

% storing the final filtered signal
wavwrite(rfinal(1:length(x)),fs,'ksfinal.wav');
clc;


% --- Outputs from this function are returned to the command line.
function varargout = untitled3_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data
% Get default command line output from handles structure
varargout{1} = handles.output;

function db_Callback(hObject, eventdata, handles)
% hObject    handle to db
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data



% --- Executes during object creation, after setting all properties.
function db_CreateFcn(hObject, eventdata, handles)
% hObject    handle to db
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function cutoff_Callback(hObject, eventdata, handles)
% hObject    handle to cutoff
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data

% --- Executes during object creation, after setting all properties.
function cutoff_CreateFcn(hObject, eventdata, handles)
% hObject    handle to cutoff
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
% called


if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
```

```
    set(hObject,'BackgroundColor','white');
end

function alpha_Callback(hObject, eventdata, handles)
% hObject    handle to alpha
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data


% --- Executes during object creation, after setting all properties.
function alpha_CreateFcn(hObject, eventdata, handles)
% hObject    handle to alpha
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function inp_Callback(hObject, eventdata, handles)
% hObject    handle to inp
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data

% --- Executes during object creation, after setting all properties.
function inp_CreateFcn(hObject, eventdata, handles)
% hObject    handle to inp
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in play1.
function play1_Callback(hObject, eventdata, handles)
% hObject    handle to play1
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data
global x
global fs
global rfinal
wavplay(rfinal(1:length(x)),fs);

% --- Executes on button press in play2.
function play2_Callback(hObject, eventdata, handles)
% hObject    handle to play2
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data
global u
global fs
```

```
wavplay(u,fs);

% --- Executes on button press in original.
function original_Callback(hObject, eventdata, handles)
% hObject    handle to original
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data
global x
global fs
wavplay(x,fs);


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global x
white = 0.1*rand(1,(length(x)));
NFFT2 = 2^nextpow2(length(white));
Y2=fft(white,NFFT2);
global fs
f2 = fs/2*linspace(0,1,NFFT2/2);
fnew2=20*log10(abs(Y2(1:length(f2))));
fnew2(1:(length(fnew2)/2))=0;
q1=ifft(Y2,NFFT2);
q2=abs(q1);
q3=max(q2);
q4=q2/q3;
global q5
q5=0.1*(real(q4(1:length(x))));
global fs
pin=str2num(get(handles.freqn,'String'));
cpnoise =0.1* cos(2*pi*pin*(0:length(x)-1)/fs)';
global u
u = x;
if (get(handles.pushbutton5,'Value') == get(hObject,'Max'))
    u = u + q5';
end
if (get(handles.pushbutton6,'Value') == get(hObject,'Max'))
    u = u + cpnoise;
end

global fs

% saving the noisy file
wavwrite(u,fs,'mynoisyfile.wav');
t=(0:length(u)-1)/fs;
set(handles.axes8,'HandleVisibility','ON');
axes(handles.axes8);
axis on;
plot(t,u/(max(u)),'Color',[0.3137 0.3137 0.3137]);
grid on;
% plotting frequecy spectrum
t=(0:length(x)-1)/fs;
L=length(x);
```

```
% fourier transform of signal
NFFT = 2^nextpow2(L);
Y=fft(u,NFFT);
f = fs/2*linspace(0,1,NFFT/2);
om22 = str2num(get(handles.order,'String'));
be = str2num(get(handles.db,'string'));
fq = str2num(get(handles.cutoff,'string'));
be2=10^(be/20);
Fs2=length(f);
temp1=(fq*Fs2)/max(f);
temp2=(pi*temp1)/Fs2;
yelpha=(cos(pi/(2*om22)))/((cos(temp2/2))*(cosh((1/om22)*acosh(be2))));
set(handles.alpha,'String',yelpha)
fnew=20*log10(abs(Y(1:length(f))));

set(handles.axes6,'HandleVisibility','ON');
axes(handles.axes6);
plot(f,fnew,'Color',[0.3137 0.3137 0.3137]);
grid on;

% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global x
global fs
pin=str2num(get(handles.freqn,'String'));
cpnoise =0.1* cos(2*pi*pin*(0:length(x)-1)/fs)';
global u
u = x;
length(u)
length(cpnoise)
if (get(handles.pushbutton6,'Value') == get(hObject,'Max'))
    u = x + cpnoise;
end
if (get(handles.pushbutton5,'Value') == get(hObject,'Max'))
    global q5
    u = u + q5';
end
global fs

% plotting frequecy spectrum
t=(0:length(x)-1)/fs;
L=length(x);

% fourier transform of signal
NFFT = 2^nextpow2(L);
Y=fft(u,NFFT);
f = fs/2*linspace(0,1,NFFT/2);
om22 = str2num(get(handles.order,'String'));
be = str2num(get(handles.db,'string'));
fq = str2num(get(handles.cutoff,'string'));
be2=10^(be/20);
Fs2=length(f);
```

```
temp1=(fq*Fs2)/max(f);
temp2=(pi*temp1)/Fs2;
yelpha=(cos(pi/(2*om22)))/((cos(temp2/2))*(cosh((1/om22)*acosh(be2))));
set(handles.alpha,'String',yelpha)
fnew=20*log10(abs(Y(1:length(f))));
set(handles.axes6,'HandleVisibility','ON');
axes(handles.axes6);
plot(f,fnew,'Color',[0.3137 0.3137 0.3137]);
grid on;

% saving the noisy file
wavwrite(u,fs,'mynoisyfile.wav');
t=(0:length(u)-1)/fs;
set(handles.axes8,'HandleVisibility','ON');
axes(handles.axes8);
axis on;
plot(t,u/(max(u)),'Color',[0.3137 0.3137 0.3137]);
grid on;




function freqn_Callback(hObject, eventdata, handles)
% hObject    handle to freqn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of freqn as text
%        str2double(get(hObject,'String')) returns contents of freqn as
a double


% --- Executes during object creation, after setting all properties.
function freqn_CreateFcn(hObject, eventdata, handles)
% hObject    handle to freqn (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns
called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

3.2 TEST APPLICATION SOURCE CODE

```
clear all;
clc;
file=input('Enter WAV filename : ','s');

% reading a wave file
[x,fs]=wavread(file);
t2=(0:length(x)-1)/fs;

%creating noise signal
noise = cos(2*pi*3*fs/8*(0:length(x)-1)/fs)';
u = x + noise;
wavwrite(u,fs,'mynoisyfile.wav');
clc;

% plot waveform
t=(0:length(u)-1)/fs;        % times of sampling instants

% Create figure
figure1 = figure('Color',[1 1 1]);

% Create axes
axes1 = axes(...
  'FontName','Arial',...
  'FontSize',9,...
  'FontWeight','bold',...
  'XColor',[0.2353 0.2353 0.2353],...
  'YColor',[0.2353 0.2353 0.2353],...
  'ZColor',[0.2353 0.2353 0.2353],...
  'Parent',figure1);
ylim(axes1,[-1 1]);
title(axes1,'Input Signal in Time Domain with Noise');
xlabel(axes1,'Time (s)');
ylabel(axes1,'Amplitude');
box(axes1,'on');
grid(axes1,'on');
hold(axes1,'all');

% Create plot
plot1 = plot(...
  t,u/max(u),...
  'Color',[0.3137 0.3137 0.3137],...
  'MarkerEdgeColor',[1 1 1],...
  'Parent',axes1);

% Create legend
legend1 = legend(...
  axes1,{'Waveform'},...
  'FontName','Arial',...
  'FontSize',9,...
  'FontWeight','bold',...
  'Orientation','horizontal',...
  'EdgeColor',[1 1 1]);
```

```
L=length(x);
NFFT = 2^nextpow2(L);
% do fourier transform of windowed signal
Y=fft(u,NFFT);

f = fs/2*linspace(0,1,NFFT/2);
fnew=20*log10(abs(Y(1:length(f))));
figure1 = figure;

% Create axes
axes1 = axes(...
  'Color',[0 1 1],...
  'FontAngle','italic',...
  'FontSize',11,...
  'FontWeight','bold',...
  'XColor',[1 0 0],...
  'YColor',[1 0 0],...
  'ZColor',[1 0 0],...
  'Parent',figure1);
xlabel(axes1,'Time (s)');
ylabel(axes1,'Amplitude');
box(axes1,'on');
grid(axes1,'on');
hold(axes1,'all');

% Create plot
plot1 = plot(...
  f,fnew,...,
  'MarkerEdgeColor',[1 1 1],...
  'Parent',axes1);

% Create legend
legend1 = legend(...
  axes1,{'Waveform'},...
  'Color',[1 1 0],...
  'FontAngle','italic',...
  'FontSize',11,...
  'FontWeight','bold',...
  'Orientation','horizontal');

legend('Spectrum');
xlabel('Frequency (Hz)');
ylabel('Magnitude (dB)');
I=ifft(Y,NFFT);

% Filter Design Starts Here
b=100;
m=6;
a=[1:m];
i=1;
alpha=1.7827;
freq=0;
Fs2=length(f);
```

```
h=1;
p=1;
omegak=((2*a-1)*pi)/(2*m);
omegam=2*acos((cos(omegak))/((alpha*(cosh(1/m*acosh(b))))));
while(freq<Fs2)
    om=2*pi*freq/(2*Fs2);
    k=exp(j*om);
    while(i<=m)
        h=h*(k-exp(j*omegam(i)));
        i=i+1;
    end
    h1(p)=abs(h);
    p=p+1;
    freq=freq+1;
    h=1;
    i=1;
end
h2=(h1/max(h1));
h3=20*log10(h2);
hold on;
plot(f,h3,'r+');
mul=(Y(1:length(f))).*h2';
mul2=abs(mul);
mulnew=20*log10(mul2);
% Create figure
figure1 = figure;

% Create axes
axes1 = axes(...
   'Color',[0 1 1],...
   'FontAngle','italic',...
   'FontSize',11,...
   'FontWeight','bold',...
   'XColor',[1 0 0],...
   'YColor',[1 0 0],...
   'ZColor',[1 0 0],...
   'Parent',figure1);
xlabel(axes1,'Time (s)');
ylabel(axes1,'Amplitude');
box(axes1,'on');
grid(axes1,'on');
hold(axes1,'all');

% Create plot
plot1 = plot(...
   f,mulnew,...
   'MarkerEdgeColor',[1 1 1],...
   'Parent',axes1);


% Create legend
legend1 = legend(...
   axes1,{'Waveform'},...
   'Color',[1 1 0],...
   'FontAngle','italic',...
   'FontSize',11,...
```

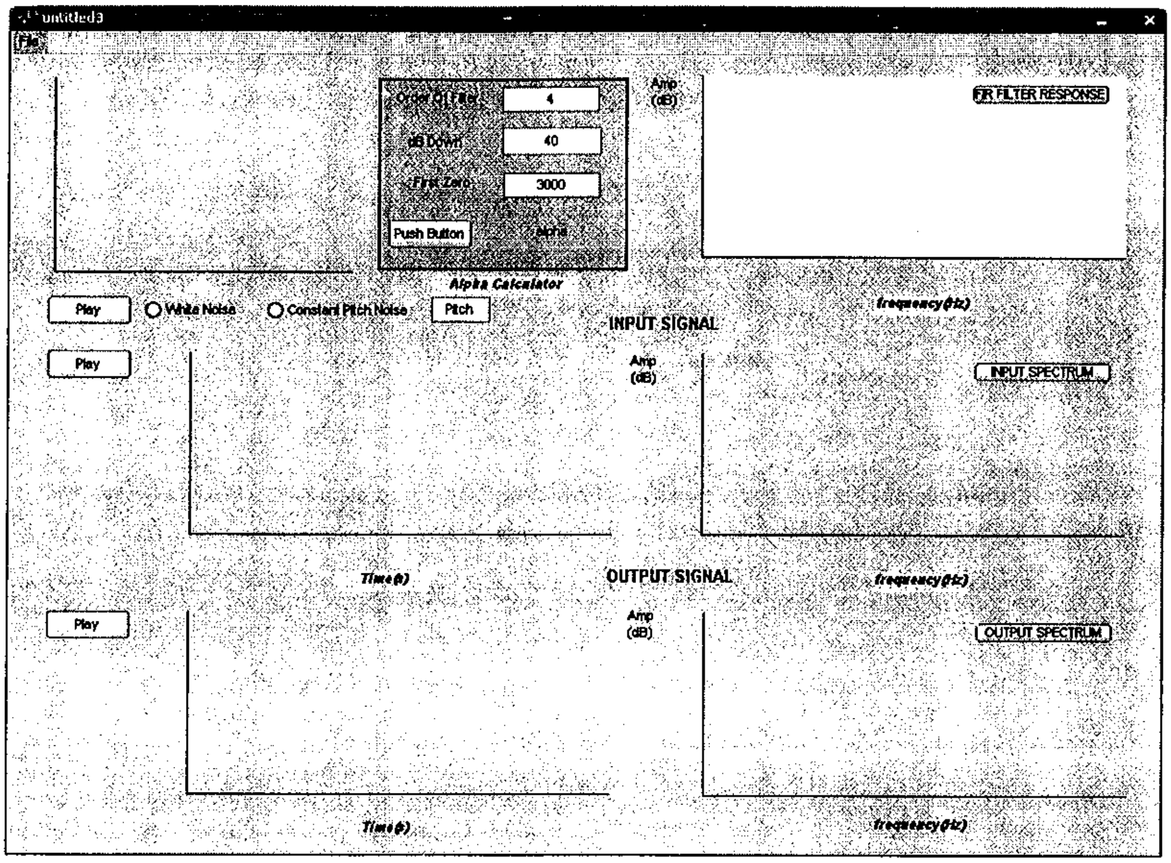
```
  'FontWeight','bold',...
  'Orientation','horizontal');
% plot(f,20*log10(mul2));
hold on;
plot(f,h3,'r+');
mul3=ifft(mul,NFFT);
mul4=abs(mul3);
mul5=max(mul4);
mul6=mul4/mul5;
% Create figure
figure1 = figure;
rmul=real(mul3(1:length(x)));
% Create axes
axes1 = axes(...
  'Color',[0 1 1],...
  'FontAngle','italic',...
  'FontSize',11,...
  'FontWeight','bold',...
  'XColor',[1 0 0],...
  'YColor',[1 0 0],...
  'ZColor',[1 0 0],...
  'Parent',figure1);
xlabel(axes1,'Time (s)');
ylabel(axes1,'Amplitude');
box(axes1,'on');
grid(axes1,'on');
hold(axes1,'all');

% Create plot
plot1 = plot(...
  t,rmul/max(rmul),...
  'MarkerEdgeColor',[1 1 1],...
  'Parent',axes1);

% Create legend
legend1 = legend(...
  axes1,{'Waveform'},...
  'Color',[1 1 0],...
  'FontAngle','italic',...
  'FontSize',11,...
  'FontWeight','bold',...
  'Orientation','horizontal');
rfinal=rmul/(max(rmul));
wavwrite(rfinal(1:length(x)),fs,'ksfinal.wav');
clc;
```

4 DEPLOYMENT

4.1 SCREEN SHOTS



Order Of Filter: 4
dB Down: 40
First Zero: 3000

Figure 36 : GUI Panel

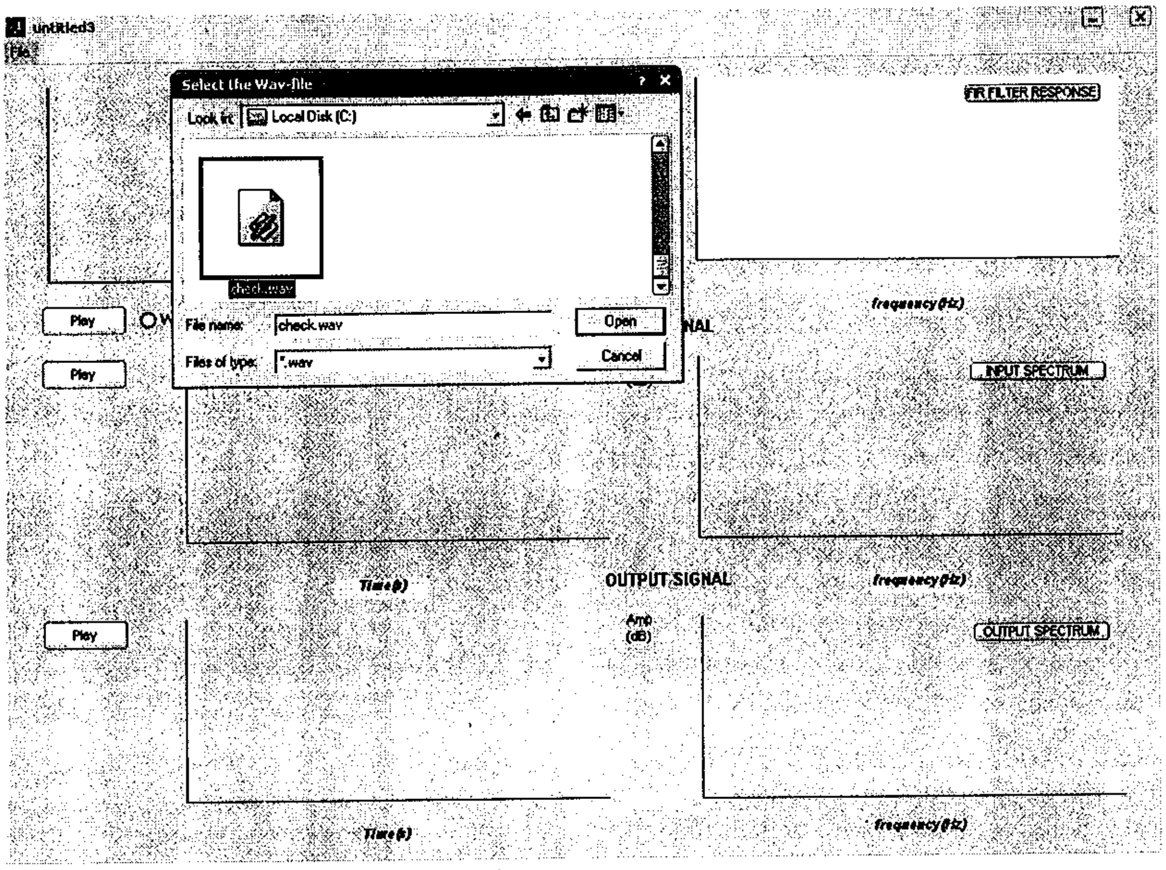
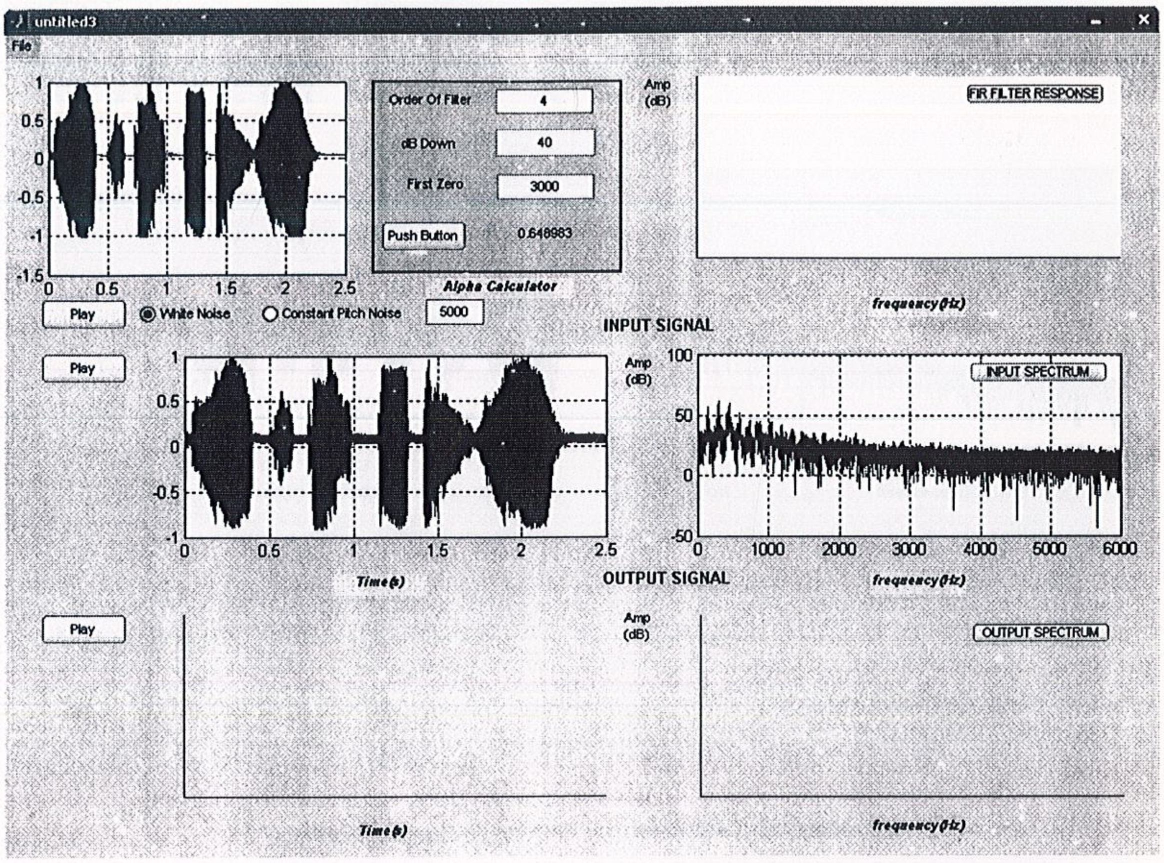
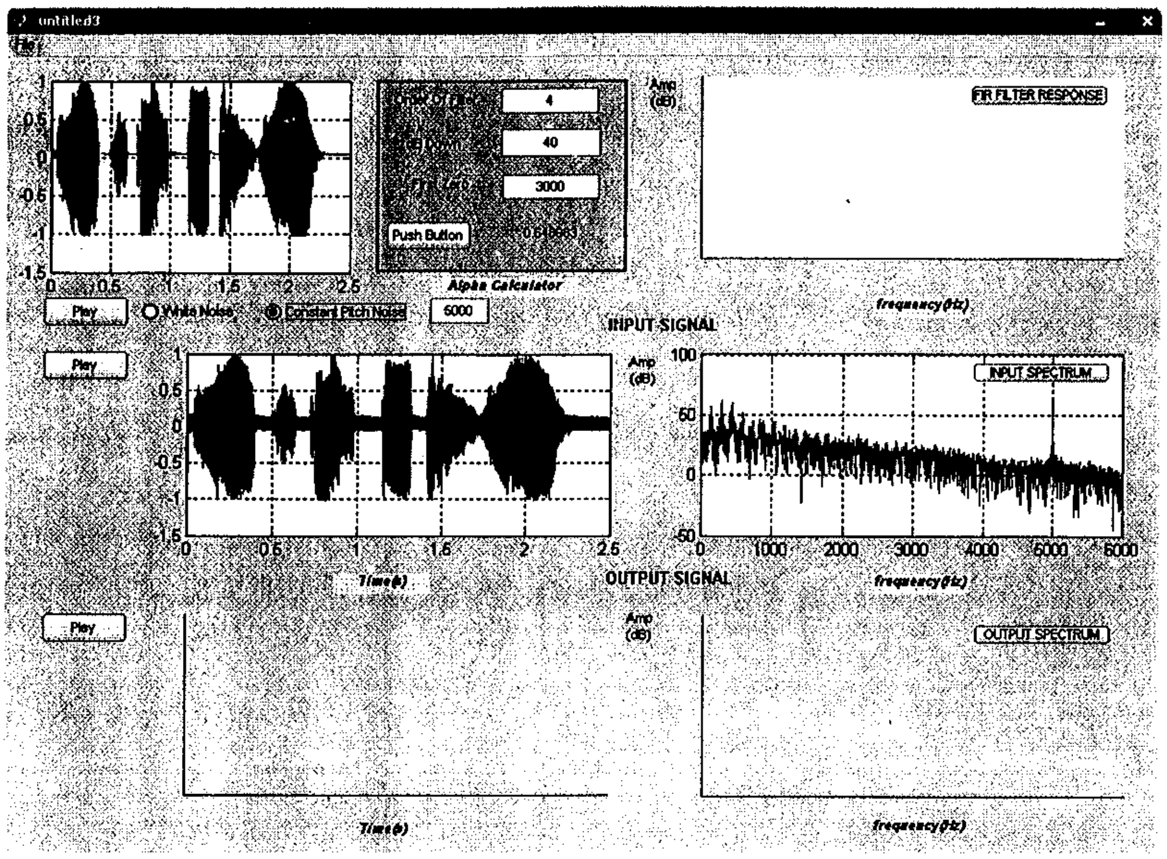


Figure 37 : Browsing an audio (wav) file



Order Of Filter: 4
dB Down: 40
First Zero: 3000
0.648983
[✓ White Noise] Constant Pitch Noise: 5000

Figure 38 : Speech signal with white noise



Order Of Filter: 4

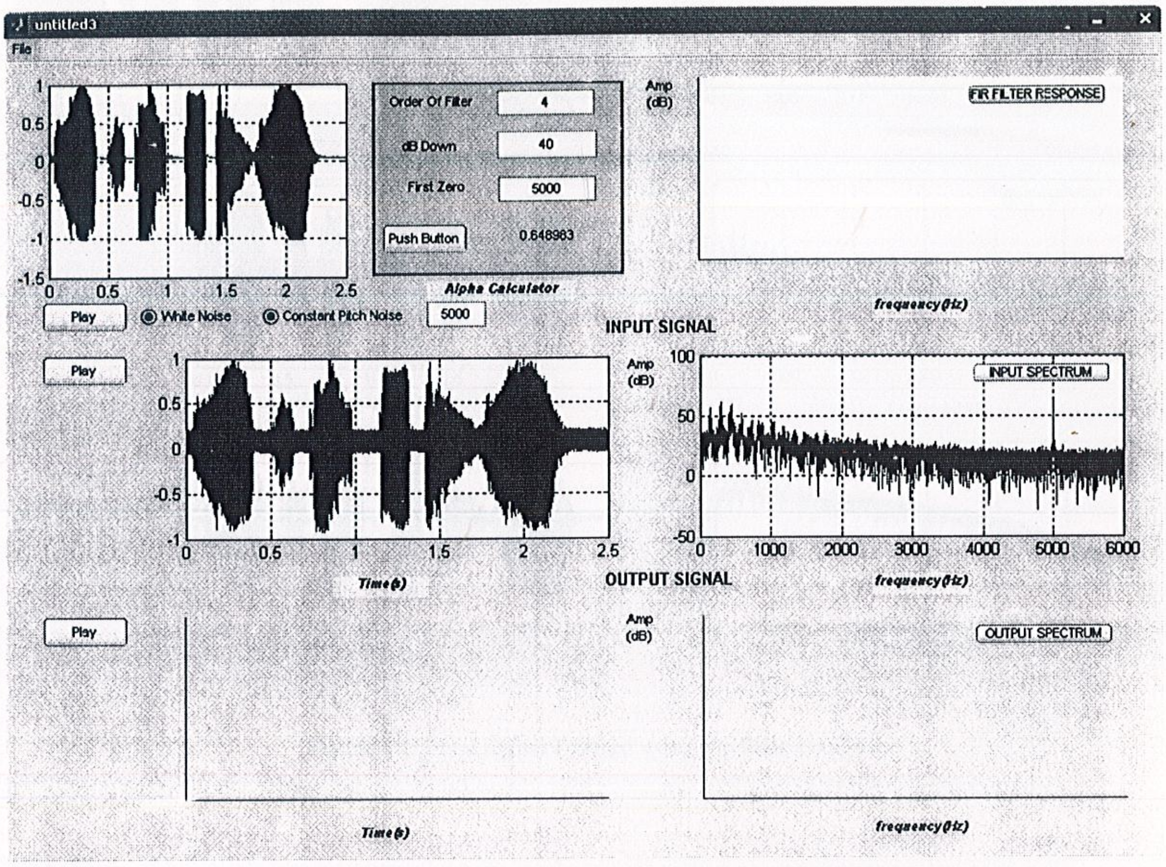
(dB) Down: 40

FIR (Hz): 3000

0.6[illegible]

○ White Noise ◉ Constant Pitch Noise 5000

Figure 39 : Speech signal with constant pitch noise at 5000 Hz



Order Of Filter: 4
dB Down: 40
First Zero: 5000
0.648983
● White Noise
● Constant Pitch Noise
5000

Figure 40 : Speech signal with white noise and constant pitch noise

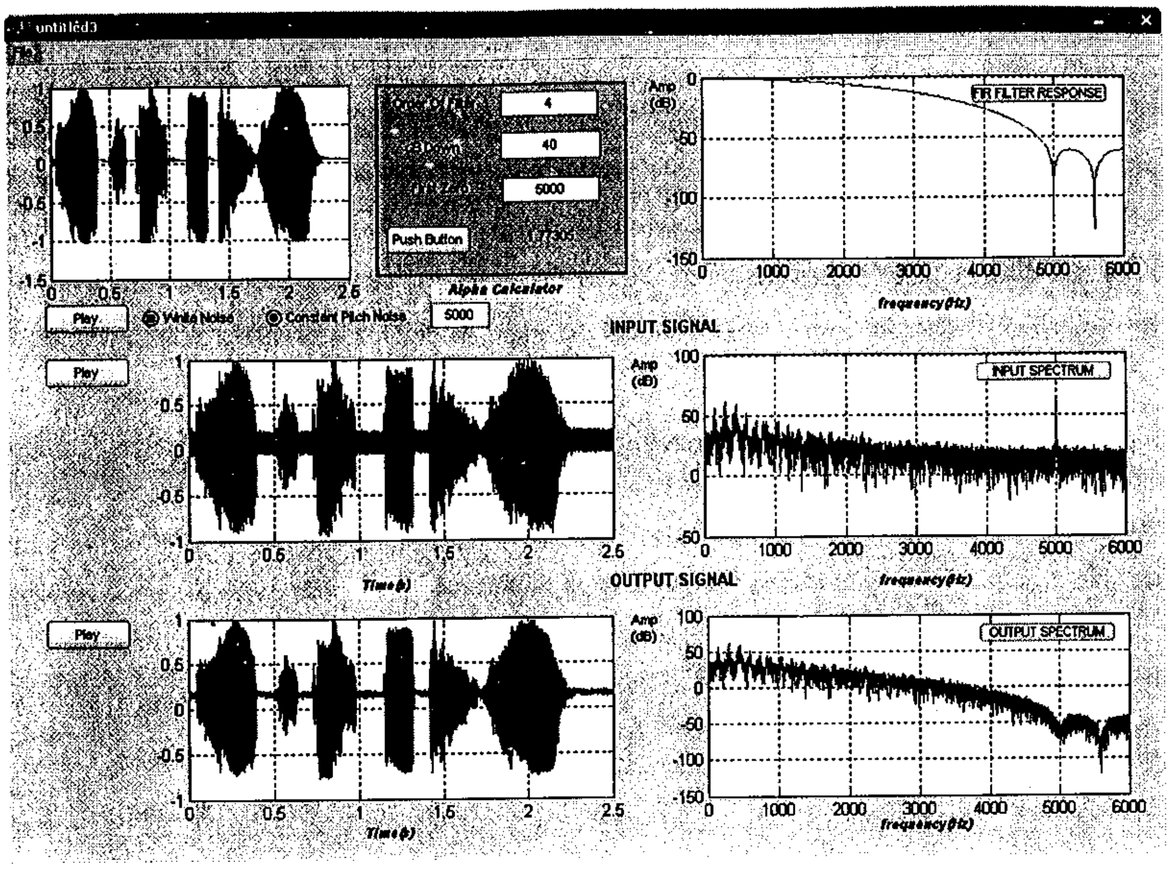


Figure 41 : Final Signal Filtered using the filter response

5 CONCLUSION

The application successfully removes the burst noise and attenuates the higher frequency components of the white noise. The filter designed here can be realised practically and can be used with applications for speech filtering. Digital filters can easily realize performance characteristics far beyond what are practically implementable with analog filters. Because of its finite impulse response characteristics it can be easily adopted for mathematical calculations. The scope of this application can be extended for various other filtering applications. The potential areas of extension are listed below

- The filter concept can be modified to give a high pass filter or band pass filter.
- Instances of constant pitch noise occurring in the voice signal can be easily removed by multiple filtering using the same concept.
- It can be used as an anti-aliasing filter.
- It can be used for 'destriping' images by subsequent low pass and high pass filtering.
- It can be used for limiting the frequency band of the luminance signal in a video recorder.

6 BIBLIOGRAPHY

Books

- Edward C. Jordan, Keith G Balmain, Electromagnetic Waves and Radiating Systems, 2nd. Ed., Prentice-Hall Inc., 1968
- John D. Kraus , Ronald J. Marhefka , Antenna for all applications, 3rd Ed., McGraw-Hill Science/Engineering/Math; (November 12, 2001)
- Alan V. Oppenheim, Ronald W. Schafer, John R. Buck, Discrete-Time Signal Processing, 2nd Ed, Prentice-Hall Inc,.2005
- B. A. Shenoi, Introduction To Digital Signal Processing And Filter Design, Wiley-Interscience (October 19, 2005)

References

- Sunil Bhooshan and Vinay Kumar. Design Of Chebyshev FIR Filter Based On Antenna Theory Approach
- L. J. Karam and J. H. McClellan. Complex Chebyshev approximations for FIR filter design. IEEE Trans. on Circuits and Systems II, 42(3):207--216, March 1995.
- X. Chen and T.W. Parks, Design of FIR Filters in the complex domain, IEEE Trans. Acoust. ,Speech, Signal Processing , Vol. ASSP35 ,pp.144-153,Feb. 1987
- D. Burnside and T.W. Parks, "Accelerated design of FIR Filter in the Complex Domain," in Proc. IEEE ICASSP 1993, pp.81-84, 1993.

Website

- http://en.wikipedia.org
- http://www.wolfram.com
- http://mathworld.wolfram.com
- http://www.dsprelated.com
- http://www.mathswork.com