Acknowledgment

This minor project is the result of a great deal of study and work, and we are much obliged to all who helped and assisted in this challenge. We are immensely grateful to Dr.G.Abhilash, Asscociate Professor ECED, who guided us in this project. This work would not have been a success without his motivation, help and suggestions. It is our great pleasure to acknowledge the role of the faculties of the department of ECE for all the knowledge they imparted through the years. Lastly, we thank our colleagues and friends for the significant exchange of knowledge which has greatly contributed to this project.

1 Abstract

The processing and transmission of large size signals under limited resources (namely power and bandwidth) is a common difficulty in the field of signal processing and communication. The solution is finding a representation of the signal in another form with a reduced number of bits and a suitable degree of accuracy, commonly known as compression. Linear predictive coding (LPC) is a method of compression wherein the system is modelled mathematically using the correlation structure of the signal. When it comes to speech, the producing system can be approximated as linear with 10 coefficients, making LPC an efficient method of compression. The quality of speech can be improved further by adding pitch and voice attributes. This project aims at the implementation of the analysis and synthesis filters in C++ using fundamental libraries and data structures.

Contents

2 Introduction

The increased popularity of digital systems in the last decade would not have been possible without the development of adequate data compression techniques. From MP3 players to cell phones, DVDs and digital television, data compression is an integral part of almost all information technology. Uncompressed audio requires substantial storage capacity. Data transfer of uncompressed audio over digital networks requires that very high bandwidth be provided for a single point to point communication. Data compression algorithms are used in these standards to reduce the number of bits required to represent an image or a video sequence or music. The use of data compression thereby reduces the bandwidth required for the transmission of the data. For a mobile user, the lower the bit rate for a voice call, the more other services (data/ image/ video) can be accommodated.

Linear prediction theory has been successfully used for the representation, modelling, compression, and computer generation of speech waveforms. Audio waveforms exhibit a high degree of continuity or sample to sample correlation i.e., the tendency of samples to be similar to their neighbours. The reason is that audio samples are digitized from continuous waveforms, and the sampling rate is usually higher than the rate needed at any particular time. To take advantage of this correlation, prior to the encoding process most audio compressors apply a pre-processing component called a predictor. The predictor coefficients are determined by minimizing the mean square error (MMSE), over a finite interval, between the actual speech samples and the linear predicted ones. The frequently used algorithms are Autocorrelation Method and the Levinson Recursion Method.

3 Theory

3.1 Digital Systems

In general, a system is an assemblage of elementary, functional, physical elements, which performs a mathematical operation on a signal. If it performs the operation on a signal which is quantized both in time and magnitude, it is a digital system. A digital system is characterized by a transfer function or equivalently by a difference equation, which mathematically describes how the system responds to any input signal. The transfer function of a linear,time-invariant(LTI) causal digital system can be represented in the Z-domain as

$$H(z) = \frac{N(z)}{D(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2} + ... + b_m z^{-m}}{a_0 + a_1 z^{-1} + a_2 z^{-2} + ... + a_n z^{-n}}$$

From the transfer function stated above, LTI digital systems can be further subdivided into two classess:

1. Finite Impulse Response System(FIR) - Impulse response is time bounded
2. Infinite Impulse Response System(IIR) - Impulse response lasts for infinite time

   The class of IIR systems can be further divided into two:

   - Auto-regressive(AR), which has all the zeros located at the origin of Z-plane
   - Auto-regressive moving average(ARMA), which has atleast one non-zero zero.

3.2 Linear Prediction

As the name suggests, linear prediction is predicting the future values of a signal as a linear function of the previous values. For such an approximations to be accurate, the signal which is being predicted should be well correlated.

Consider a single pole IIR system which is excited by an input sequence $x[n]$ giving an output sequence $y[n]$, whose transfer function is

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 + a_1 z^{-1}}$$

In time domain, the corresponding difference equation can be written as

$$y[n] = -a_1 y[n-1] + x[n] \tag{1}$$

This equation states that the sample value can be estimated as a function of one previous sample value. Let $\hat{y}[n]$ be the estimate of $y[n]$ and $e[n]$ be the error in the estimation.

$$\hat{y}[n] = -b_1 y[n-1]$$
$$y[n] = \hat{y}[n] + e[n]$$

$$y[n] = -b_1 y[n-1] + e[n] \tag{2}$$

Eq.(1) and Eq.(2) looks analogous where b_1 and $e[n]$ are at the equivalent positions of a_1 and $x[n]$. This shows that even though this is an estimation problem, essentially it models the system from the signal. For the optimum estimation, the energy of error signal $e[n]$ has to be minimized over the coefficient b_1. i.e,

$$e[n] = y[n] - \hat{y}[n]$$
$$e[n] = y[n] + b_1 y[n-1] \tag{3}$$

Differentiating $E(e[n]^2)$,

$$\begin{aligned}\frac{d}{db_1}E\{e[n]^2\} &= E\{\frac{d}{db_1}e[n]^2\} \\ &= E\{2e[n]\frac{d}{b_1}e[n]\} \\ &= 2E\{e[n]y[n-1]\} \\ &= 2E\{((y[n]+b_1y[n-1])y[n-1])\}\end{aligned}$$

Equating to zero,

$$E\{y[n]y[n-1]+b_1y[n-1]y[n-1]\}=0$$

i.e,

$$R_y(-1)+b_1R_y(0)=0$$

Where $R_y[n]$ be the correlation coefficients of the signal $y[n]$. Since the process is stationary, $R_y(-n)=R_y(n)$

$$b_1=\frac{-R_y(1)}{R_y(0)} \tag{4}$$

Eq.(4) gives the optimum coefficient for the prediction. Hence from Eq.(3) the prediction error filter for a single pole IIR system can be realized as shown in Figure 1.A prediction error filter essentially removes the correlation content of the signal. Thus the output of prediction error filter is likely to be white noise, since the autocorrelation approaches to delta function. This implicitly says that the original signal can be reconstructed by exciting the inverse system of prediction error filter(synthesis filter) with the error signal. Even a white noise having the same 1^{st} and 2^{nd} order moments of the prediction error can reproduce the original signal with a tolerable error. The synthesis filter of a single pole IIR system is shown in figure 2.

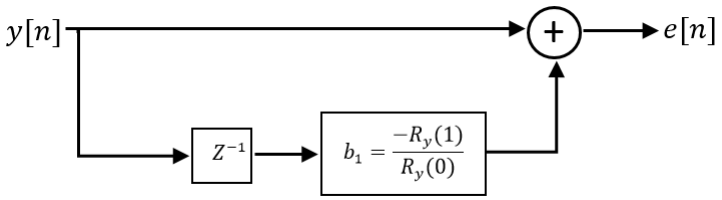

Figure 1: Prediction error filter of a single pole IIR system

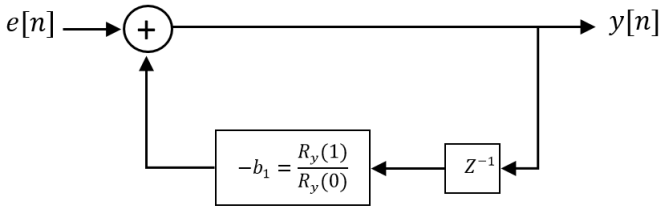

Figure 2: Synthesis filter of a single pole IIR system

Consider a general IIR system with k number of poles and all zeros located at the origin. Let $x[n]$ be the input to the system and $y[n]$ be the output. The transfer function of the system can be written as

$$H(z) = \frac{Y(z)}{X(z)} = \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2} + ... + a_k z^{-k}}$$

The corresponding difference equation is,

$$y[n] = -a_1 y[n-1] - a_2 y[n-2] - a_3 y[n-3] - ... + x[n]$$

Let $\hat{y}[n]$ be an estimate of $y[n]$ from k previous samples with an error of e[n]. i.e,

$$\hat{y}[n] = \sum_{i=1}^{k} -b_i y[n-i] \tag{5}$$

$$e[n] = y[n] + \sum_{i=1}^{k} b_i y[n-i] \tag{6}$$

For the prediction to be optimum, the coefficients should be chosen in such a way that the energy of the error part becomes as minimum as possibble.

Thus, differentiating $E\{e[n]^2\}$ with respect to b_i, $i = 1, 2, 3, ..., k$

$$\begin{aligned}\frac{d}{db_i}E\{e[n]^2\} &= E\{\frac{d}{db_i}e[n]^2\} \\ &= E\{2e[n]\frac{d}{db_i}e[n]\} \\ &= E\{2e[n]y[n-i]\}\end{aligned}$$

Equating to zero,

$$E\{e[n]y[n-i]\} = 0 \tag{7}$$

Visualising the estimate $\hat{y}[n]$ as a plane spanned by k previous samples $y[n-1], y[n-2], y[n-3], ..., y[n-k]$, Eq.(7) says that the error e[n] should be orthogonal to this plane to make the prediction optimum.

$$E\{y[n-i](y[n] + \sum_{j=1}^{k} b_j y[n-j])\} = 0$$

$$R_y(i) + \sum_{j=1}^{k} b_j R_y(i-j) = 0, \qquad i = 1, 2, 3, ..., k \tag{8}$$

The energy of error signal $e[n]$ is given by

$$\begin{aligned}E\{e[n]^2\} &= E\{e[n](y[n] + b_1 y[n-1] + b_2 y[n-2] + ... + b_k y[n-k]\} \\ &= E\{e[n]y[n]\} \\ &= E\{(y[n] + b_1 y[n-1] + b_2 y[n-2] + ... + b_k y[n-k])y[n]\} \\ &= R_y(0) + b_1 R_y(1) + b_2 R_y(2) + ... + b_k R_y(k)\end{aligned}$$

This equation together with Eq.(8) gives a consistent system of linear equation as follows,

$$\begin{aligned}R_y(0) + b_1 R_y(1) + b_2 R_y(2) + ... + b_k R_y(k) &= 0 \\ R_y(1) + b_1 R_y(0) + b_2 R_y(1) + ... + b_k R_y(k-1) &= 0 \\ R_y(2) + b_1 R_y(1) + b_2 R_y(0) + ... + b_k R_y(k-2) &= 0 \\ R_y(3) + b_1 R_y(2) + b_2 R_y(1) + ... + b_k R_y(k-3) &= 0 \\ . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\ . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\ . \quad . \quad . \quad . \quad . \quad . \quad . \quad . \\ R_y(k) + b_1 R_y(k-1) + b_2 R_y(k-2) + ... + b_k R_y(0) &= 0\end{aligned}$$

The matrix representation is given as,

$$\begin{bmatrix} R_y(0) & R_y(1) & R_y(2) & ... & R_y(k) \\ R_y(1) & R_y(0) & R_y(1) & ... & R_y(k-1) \\ R_y(2) & R_y(1) & R_y(0) & ... & R_y(k-2) \\ . & . & . & ... & . \\ . & . & . & ... & . \\ . & . & . & ... & . \\ R_y(k) & R_y(k-1) & R_y(k-2) & ... & R_y(0) \end{bmatrix} \begin{bmatrix} 1 \\ b_1 \\ b_2 \\ . \\ . \\ . \\ b_k \end{bmatrix} = \begin{bmatrix} \varepsilon_{min} \\ 0 \\ 0 \\ . \\ . \\ . \\ 0 \end{bmatrix}$$

$$R_{k\times k} \times b_{k\times 1} = \varepsilon_{k\times 1} \tag{9}$$

$$b_{k\times 1} = R^{-1}_{k\times k} \times \varepsilon_{k\times 1} \tag{10}$$

Eq.(10) gives the optimum prediction coefficients with least possible error energy. The prediction error filter and synthesis filter can be realised in the same way as shown in Figure. 1 and Figure 2 respectively.

The auto-correlation method used involves computation of inverse of the correlation matrix. This process demands a high computational gravity (techniques for calculation of inverse typically have a complexity of $\emptyset(n^{2.373-3})$) thereby greatly increasing the execution time of the algorithm, bringing into question its relevance in real-time application. This necessitates the search for an alternate algorithm which extracts the IIR filter coefficients more efficiently.

A simple solution one can think of is an algorithm which computes the predictor coefficients recursively. In other words, an algorithm which calculates the p-tap predictor coefficients from $p-1$-tap predictor. Suppose we have a p-tap predictor with the optimum set of coefficients. Let $y[n]$ be the signal, $\hat{y}_p[n]$ be the predicted signal and $e_p[n]$ be the prediction error.

$$\hat{y}_p[n] = -a_{p1}y[n-1] - a_{p2}y[n-2] - a_{p3}y[n-3] - ... - a_{pp}y[n-p]$$

$$e_p[n] = y[n] + a_{p1}y[n-1] + a_{p2}y[n-2] + a_{p3}y[n-3] + ... + a_{pp}y[n-p]$$

The predicted signal $\hat{y}_p[n]$ lies in a space spanned by the past p samples. Thus for the optimum predictor with minimum error variance, the prediction error $e[n]$ should be orthogonal to the space spanned by the past p samples. This notion can be written as

$$E\{e_p[n]y[n-k]\} = 0 \quad , \quad 1 \leq k \leq p$$

This intuition can be used to define a function $g_p[k]$ as

$$g_p[k] = E\{e_p[n]y[n-k]\} \tag{11}$$

which is termed as gap function of order p. Let $g_{p+1}[k]$ be the gap function of $(p+1)$ order predictor whose coefficients are not yet revealed. Finding the $(p+1)$ order predictor from p order predictor is equivalent to generating $g_{p+1}[k]$ from $g_p[k]$. The only requirement on $g_{p+1}[k]$ is the property of 'gap'. i.e,

$$g_{p+1}[k] = 0 \quad , \quad 1 \leq k \leq p+1$$

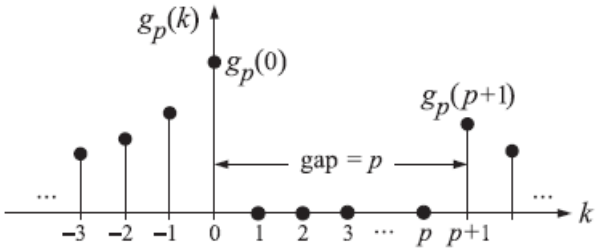

Figure 3: $g_p[k]$

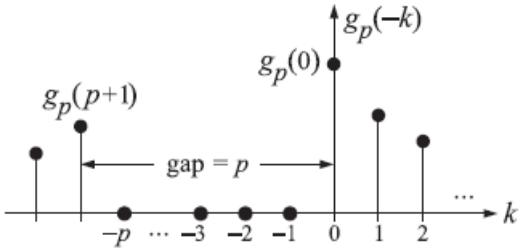

Figure 4: $g_p[-k]$

Considering reversing the gap function $g_p[k]$, one solution is to shift the reversed gap function $g_p[-k]$ by $p+1$ indices towards the right, adding it to the gap function $g_p[k]$ by scaling in such a way that the resulting $g_{p+1}[k]$ has a zero at $k = p+1$. During this process, an additional zero is introduced at $k = p+1$ while conserving all the p zeroes in $g_p[k]$.

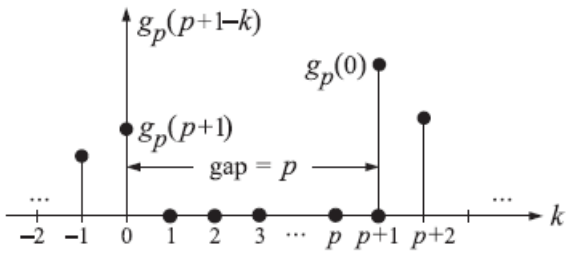

Figure 5: $g_p[p-k+1]$

$$g_{p+1}[k] = g_p[k] - \gamma_{p+1} g_p[p-k+1] \tag{12}$$

where γ_{p+1} is given by,

$$\gamma_{p+1} = \frac{g_p[p+1]}{g_p[0]} \tag{13}$$

Consider the error variance of p order predictor,

$$\begin{aligned} E_p &= E\{e_p[n]^2\} \\ &= E\{e_p[n]y[n-0]\} \\ &= g_p[0] \end{aligned} \tag{14}$$

Similarly, the error variance of the $(p+1)$ filter is given by,

$$\begin{aligned} E_{p+1} &= g_{p+1}[0] \\ &= g_p[0] - \gamma_{p+1} g_p[p+1] \\ &= g_p[0] - \gamma_{p+1}^2 g_p[0] \\ &= (1 - \gamma_{p+1}^2) g_p[0] \end{aligned} \tag{15}$$

which makes it quite obvious that the strength of the error goes on decreasing for increasing predictor order.

Now, the gap function $g_{p+1}[k]$ is derived from $g_p[k]$. Similarly the predictor coefficients also have to be derived from that of the previous predictor. Again, consider the definition of gap function $g_p[k]$,

$$\begin{aligned} g_p[k] &= E\{e_p[n]y[n-k]\} \\ &= E\{y[n-k]\sum_{i=0}^{p} a_{pi} y[n-i]\} \\ &= \sum_{i=0}^{p} a_{pi} R[k-i] \end{aligned} \tag{16}$$

which represents the convolution of predictor coefficients and correlation coefficients. Taking into Z-domain, Eq.(16) becomes

$$G_p(z) = A_p(z)S_{yy}(z) \tag{17}$$

$$G_{p+1}(z) = A_{p+1}(z)S_{yy}(z) \tag{18}$$

Where $A_p(z)$ denotes the z-transform of prediction error filter of order p and $S_{yy}(z)$ denotes the power-spectral density(PSD) of the input signal.

Going back to Eq.(12) and taking the z-transform,

$$G_{p+1}(z) = G_p(z) - \gamma_{p+1}z^{-(p+1)}G_p(z^{-1})$$

substituting Eq.(17) and Eq.(18),

$$A_{p+1}(z)S_{yy}(z) = A_p(z)S_{yy}(z) - \gamma_{p+1}z^{-(p+1)}A_p(z^{-1})S_{yy}(z^{-1})$$

Since y[n] is a stationary process, $R_{yy}[n] = R_{yy}[-n]$, hence $S_{yy}(z) = S_{yy}(z^{-1})$. Thus the equation gets simplified to,

$$A_{p+1}(z) = A_p(z) - \gamma_{p+1}z^{-(p+1)}A_p(z^{-1}) \tag{19}$$

Taking the inverse z-transform,

$$\begin{bmatrix} 1 \\ a_{p+1,1} \\ a_{p+1,2} \\ a_{p+1,3} \\ . \\ . \\ . \\ a_{p+1,p+1} \end{bmatrix} = \begin{bmatrix} 1 \\ a_{p,1} \\ a_{p,2} \\ a_{p,3} \\ . \\ . \\ . \\ 0 \end{bmatrix} - \gamma_{p+1} \begin{bmatrix} 0 \\ a_{p,p} \\ a_{p,p-1} \\ a_{p,p-2} \\ . \\ . \\ . \\ 1 \end{bmatrix} \tag{20}$$

This efficient algorithm is called **Levinson's Recursion Algorithm**. Considering the reverse polynomial $A_p^R(z) = z^{-p}A_p(z^{-1})$, Eq.(19) can be written as

$$A_{p+1}(z) = A_p(z) - \gamma_{p+1}z^{-1}A_p^R(z) \tag{21}$$

Replacing z by z^{-1} in Eq.(19),

$$A_{p+1}(z^{-1}) = A_p(z^{-1}) - \gamma_{p+1}z^{p+1}A_p(z)$$

$$z^{-(p+1)}A_{p+1}(z^{-1}) = z^{-(p+1)}A_p(z^{-1}) - \gamma_{p+1}A_p(z)$$

$$A_{p+1}^R(z) = z^{-1}A_p^R(z) - \gamma_{p+1}A_p(z) \tag{22}$$

Eq.(21) and Eq.(22) can be combined together as,

$$\begin{bmatrix} A_{p+1}(z) \\ A_{p+1}^R(z) \end{bmatrix} = \begin{bmatrix} 1 & -\gamma_{p+1}z^{-1} \\ -\gamma_{p+1} & z^{-1} \end{bmatrix} \begin{bmatrix} A_p(z) \\ A_p^R(z) \end{bmatrix} \tag{23}$$

which represents the forward recursion. The interpretation of the inverse z-transform of this matrix leads to a lattice form realization of computing filter coefficients as shown below

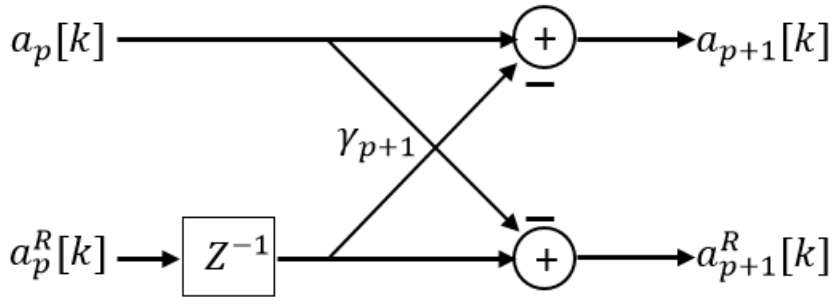

Figure 6: Computing predictor coefficients using lattice structure

Multiplying the forward recursion equation - Eq.(23) by $Y[z]$

$$\begin{bmatrix} A_{p+1}(z) \\ A_{p+1}^R(z) \end{bmatrix} [Y[z]] = \begin{bmatrix} 1 & -\gamma_{p+1}z^{-1} \\ -\gamma_{p+1} & z^{-1} \end{bmatrix} \begin{bmatrix} A_p(z) \\ A_p^R(z) \end{bmatrix} [Y[z]]$$

$$\begin{bmatrix} A_{p+1}(z)Y[z] \\ A_{p+1}^R(z)Y[z] \end{bmatrix} = \begin{bmatrix} 1 & -\gamma_{p+1}z^{-1} \\ -\gamma_{p+1} & z^{-1} \end{bmatrix} \begin{bmatrix} A_p(z)Y[z] \\ A_p^R(z)Y[z] \end{bmatrix}$$

Since $A_p(z)Y[z] = E_p(z)$ (which can be denoted as $E_p^+(z)$ indicating the forward prediction error), $A_p^R(z)Y[z]$ can be denoted as $E_p^-(z)$. Thus,

$$\begin{bmatrix} E_{p+1}^+(z) \\ E_{p+1}^-(z) \end{bmatrix} = \begin{bmatrix} 1 & -\gamma_{p+1}z^{-1} \\ -\gamma_{p+1} & z^{-1} \end{bmatrix} \begin{bmatrix} E_p^+(z) \\ E_p^-(z) \end{bmatrix} \tag{24}$$

Interpreting the inverse z-transform of this matrix equation yields to the lattice realization of prediction error filter which is given below.

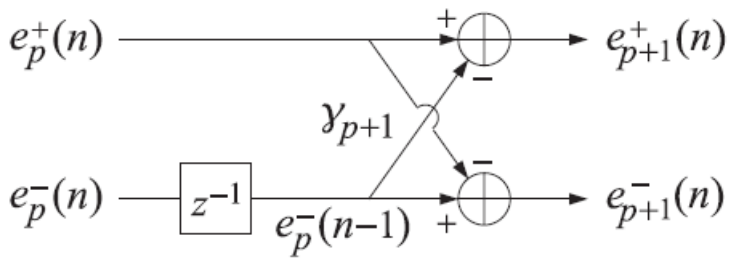

Figure 7: A single lattice section of prediction error filter

For $p = 0$, $a_0 = [1, 0]$, hence $e_0^+[n] = e_0^-[n] = y[n]$. Thus the complete prediction error filter is realized using lattice structure as shown below.

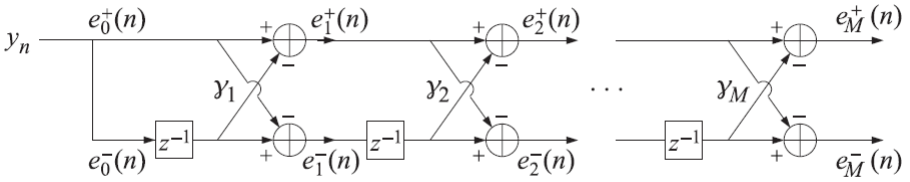

Figure 8: Lattice realization of prediction error filter

3.3 Speech Modelling

The human process of speech production involves three major levels of processing. The first being the intention to speak including formation of a thought. The second level includes the conversion of abstract thought into a permissible language. Finally comes the articulation of the message by the coordination of lungs, glottis, larynx, tongue, lips, jaw and other parts of the speech producing system.

3.4 Speech Coding

Speech is a very special type of signal for many reasons. Most importantly its non-stationary nature makes it hard to analyse and model. Furthermore, factors like intelligibility, coherence and other such characteristics play a vital role in the analysis of the speech signals. Also, from the communication point of view, the minimum required sampling rate is 8kHz, where each sample needs at least 8 bits for a toll quality voice. As bandwidth is the parameter which affects the cost of processing, speech signals should be subjected to compression before transmission.Speech coding is the process

of obtaining a compact representation of voice signals for efficient transmission over band limited wired and wireless channels. Speech coders have become essential components in telecommunications and in the multimedia infrastructure. It involves creating a minimally redundant representation of the speech signal that can be efficiently transmitted or stored in digital media, and decoding the signal with the best quality. Hence the desirable properties of the speech coder includes a low bit-rate while maintaining a minimal quality for interpretation.

The human speech producing system can be modelled as a linear time-variant system. The nature of the system varies when it produces different syllables or sounds. Still, it can be modelled as an LTI system over a short interval of time of each syllable. Typically this duration is taken as 22.5 milli-second. For a speech signal sampled at 8kHz, it can be modelled as an Auto-regressive system with 10 poles, over this duration. Using linear prediction, this system is well approximated by 10 coefficients for each frame of 22.5 milli-second. The signal is reconstructed by exciting the estimated system with a locally generated noise signal with the same variance as of the prediction error. Thus the data rate required reduces to 3.55kbit/s from 64kbit/s!. The quality of speech can be further improved by considering the pitch and voiced/unvoiced attributes of the sound.

4 Results

4.1 Modelling Auto-regressive System using Linear Prediction

Design of an auto-regressive IIR system of six poles is started from the complex plane by setting the poles. Poles are taken as conjugate pairs to keep the system real. The frequency response of the system is computed by the evaulation of transfer function along the unit-circle in the Z-domain. The filter coefficients are figured out from the poles by polynomial multiplication in Z-domain. Then the filter structure is realised and the system is excited by a gaussian noise. The linear predictive analysis is done on the output signal and the filter coefficients are estimated. The impulse response is found with the actual cofficients and estimated coefficients. The magnitude and phase responses of the estimated system is also found. The filter coefficients,impulse response and frequency response of actual system and estimated system are plotted together and compared.

Coefficients of Actual System	Estimated coefficients
2.0161	1.9605
2.0801	2.0413
1.2941	1.3231
0.5200	0.5976
0.1260	0.1747
0.0156	0.0294

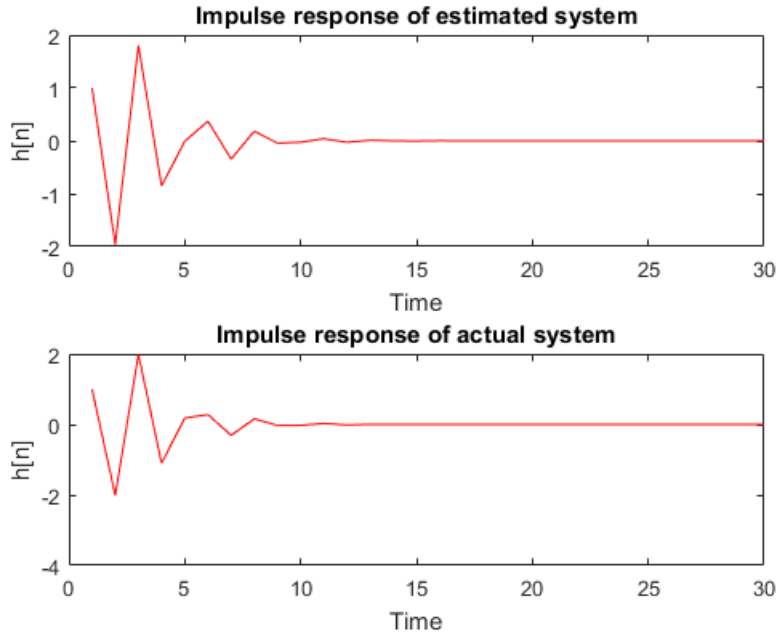

Figure 9: Impulse Response Comparison

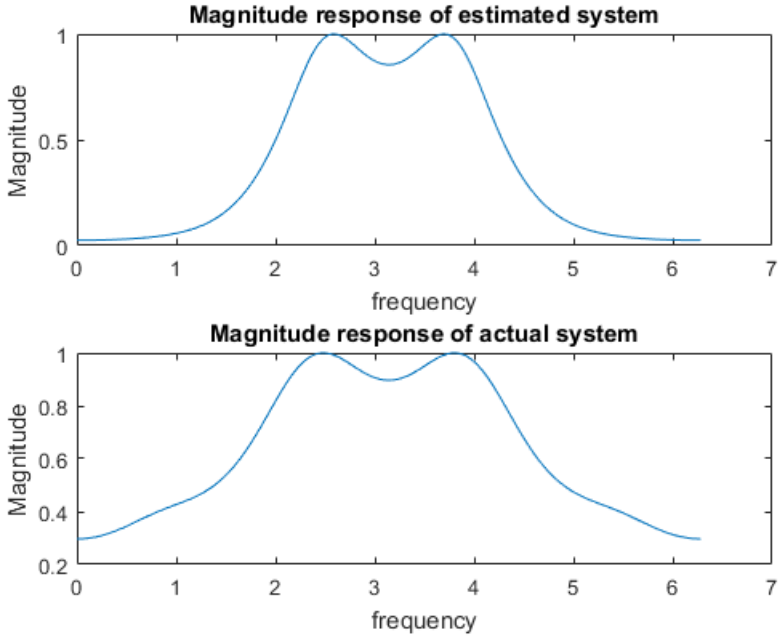

Figure 10: Magnitude Spectrum Comparison

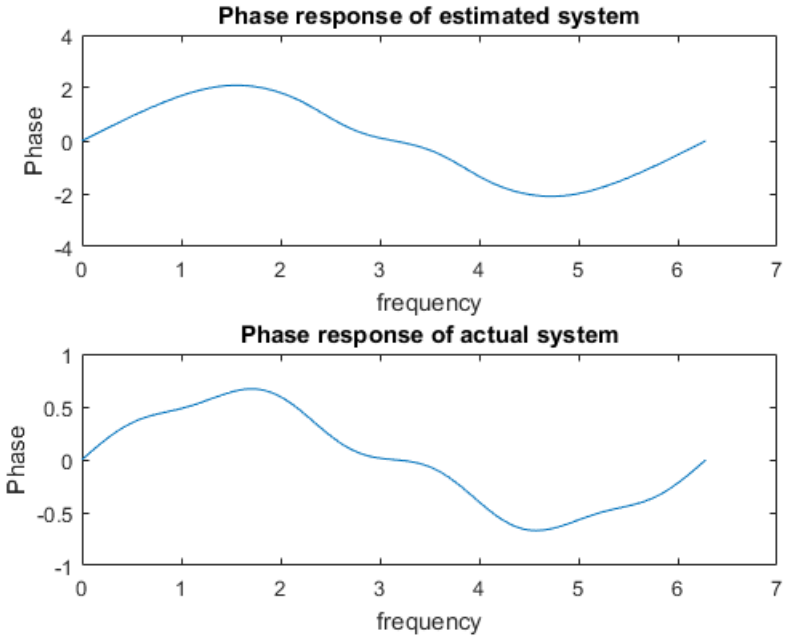

Figure 11: Phase Spectrum Comparison

4.2 Speech Coding using linear prediction

4.2.1 Implementation in Matlab

A WAV file of speech signal sampled at 8 kHz is loaded to an array. The signal is splitted into frames of duration 22.5 milli-second. Linear predictive analysis is done on each frame and the estimated coefficients are stored along with the error variance. The analysis is done using both the algorithms mentioned in the theory section. In the reconstruction part, a white gaussian noise with the same variance and zero mean is generated. This noise is used to excite the system modelled using estimated coefficients and corresponding frame is reconstructed as the output of the system with permissible error. The process is done for each frame and they are concatenated to give the complete speech signal. The reconstructed signal retains the information content of the original speech signal to an appreciable extend.

The first method of analysis, namely autocorrelation method, is implemented by realizing the correlation matrix as a multidimensional array of order $(k+1) \times (k+1)$, where k is the order of decorrelating filter realized. The filter coefficients are obtained using Eq.(10). The prediction error filter(FIR) and synthesis filter(IIR) are realized in direct form similar to Figure 1 and Figure 2 Respectively.

The analysis is repeated using Lenvinson's Recursion algorithm. The computation of coefficients is done using lattice sections which is computationally way simpler compared to the autocorrelation method, since the later involves computation of matrix inverse. The prediction error filter is also realized in lattice form. The coefficient array $A_p[k]$, reversed coefficient array $A_p^R[k]$, residues $e_p^+[n]$ and $e_p^-[n]$ are realized as arrays of type double.

The result obtained through both the methods were exactly identical. Taking an arbitiary segment of the input audio signal, the autocorrelation function of original signal, decorrelated signal and reconstructed signal are plotted and compared. The power-spectral-desnsity of the same set of signals were also studied. All of the mentioned plots are given below.

4.2.2 Comparison of Autocorrelation Functions

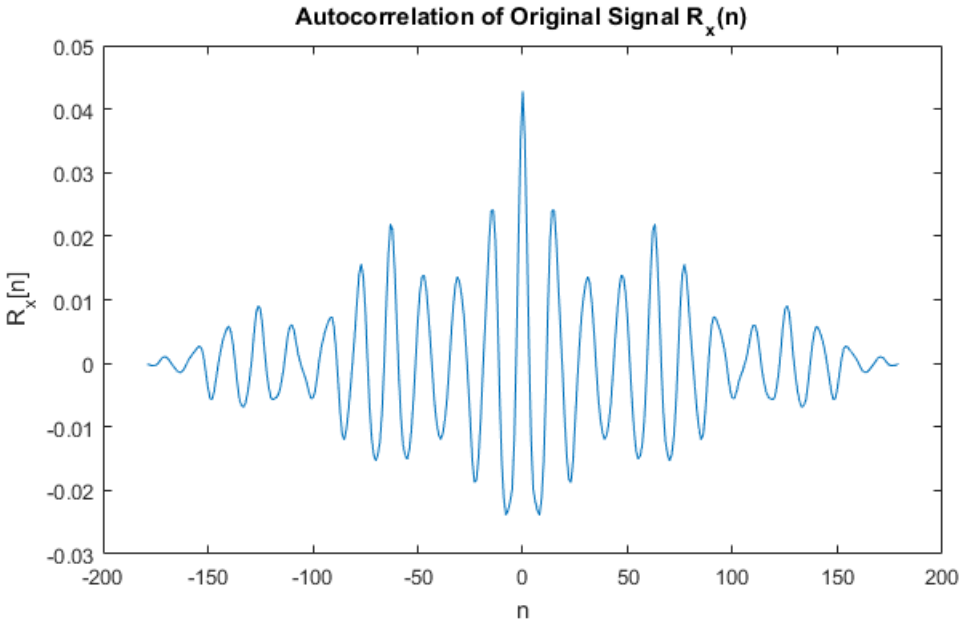

Figure 12: Autocorrelation function of original segment

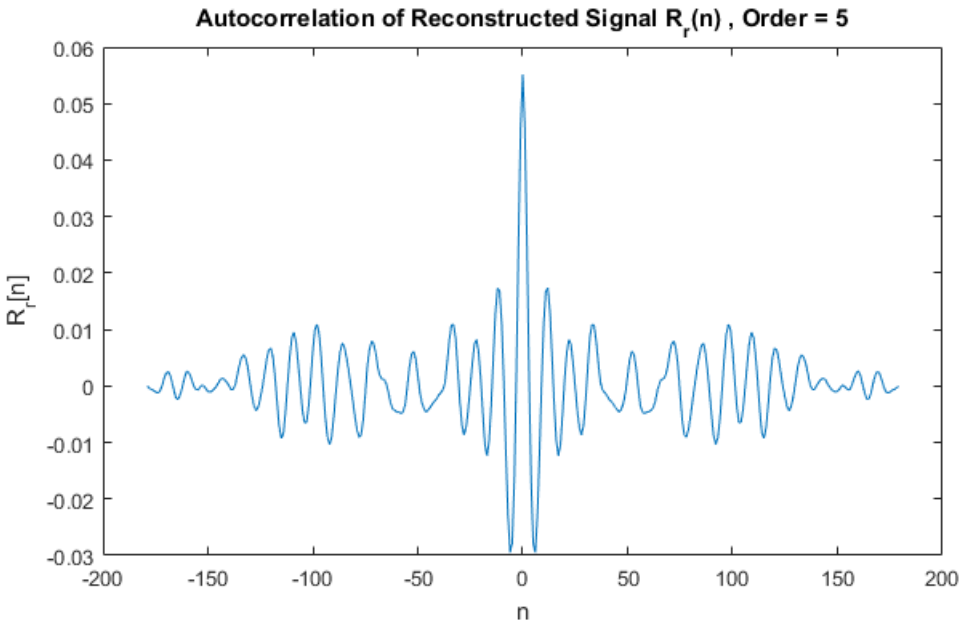

Figure 13: Autocorrelation function of reconstructed segment with 5 coefficients

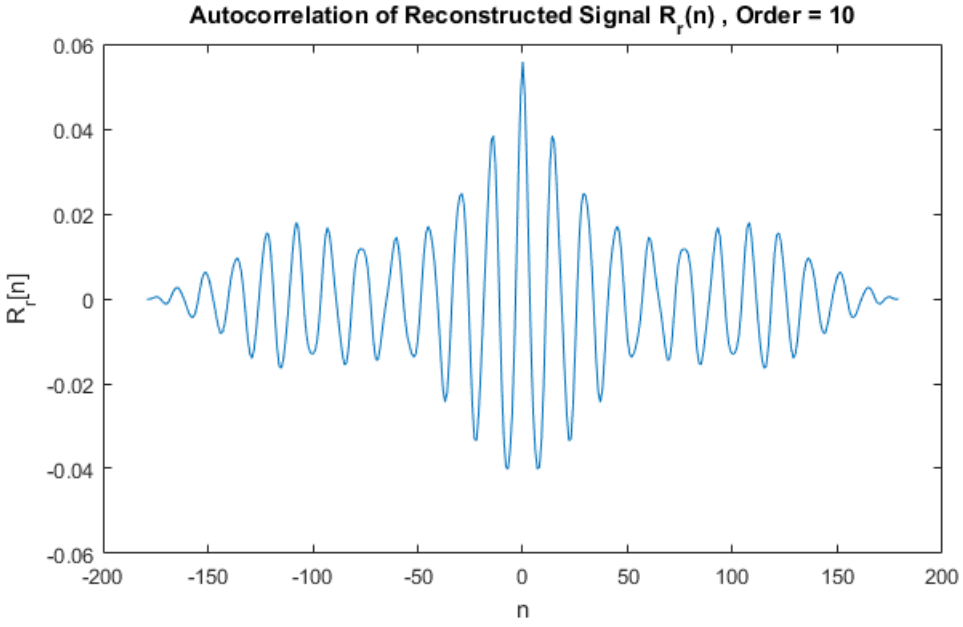

Figure 14: Autocorrelation function of reconstructed segment with 10 coefficients

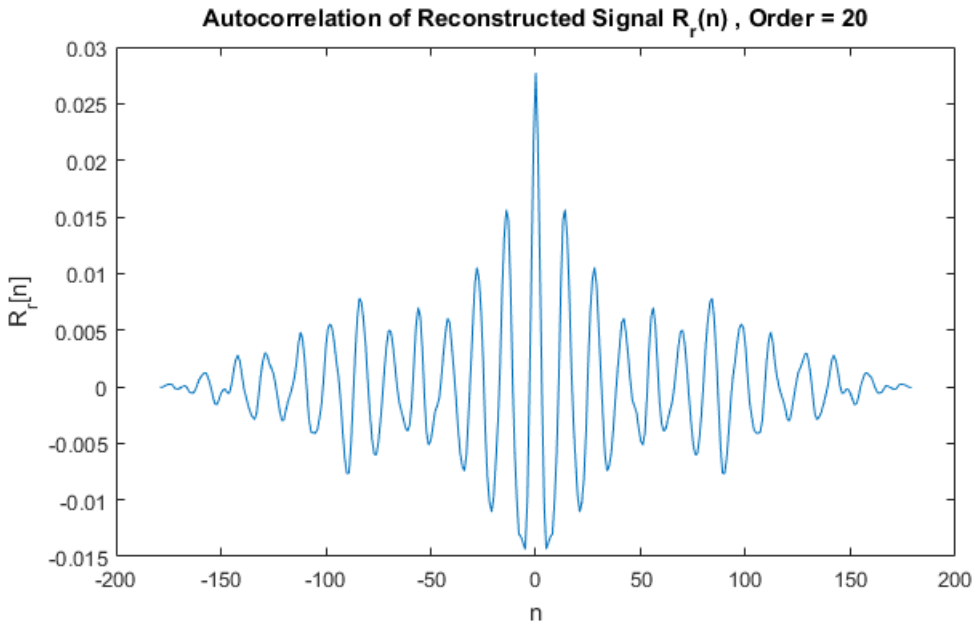

Figure 15: Autocorrelation function of reconstructed segment with 20 coefficients

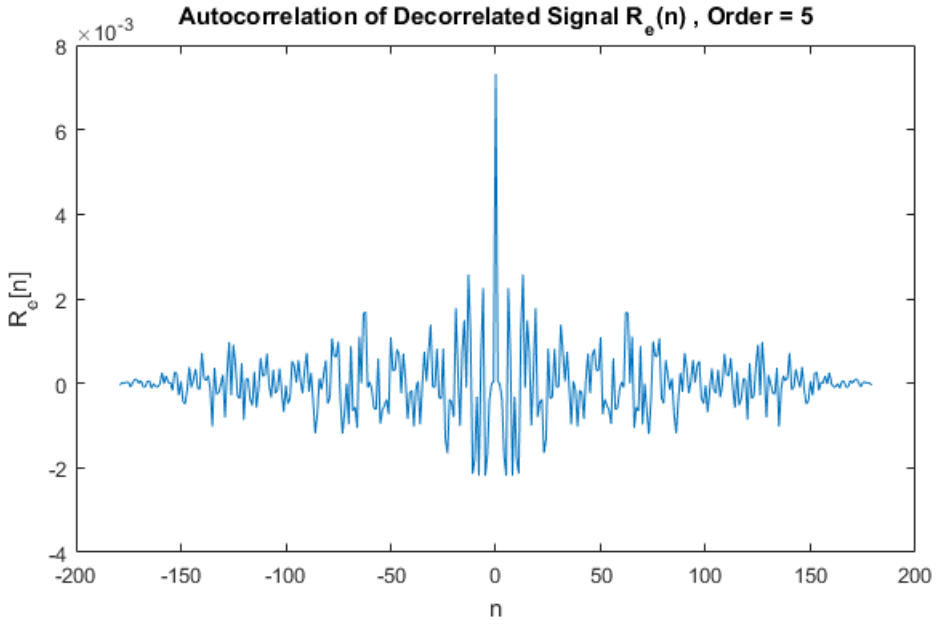

Figure 16: Autocorrelation function of decorrelated segment with 5 coefficients

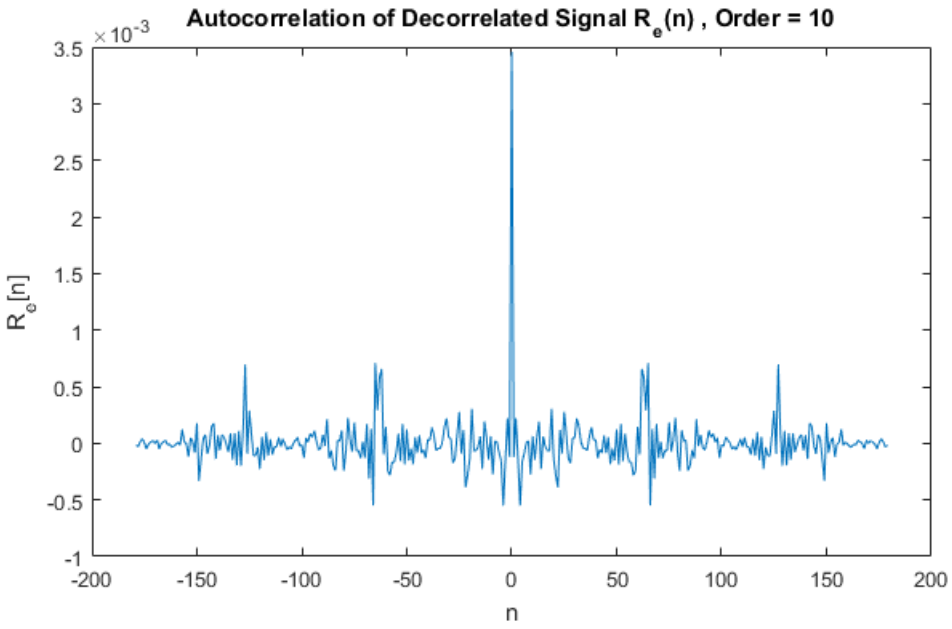

Figure 17: Autocorrelation function of decorrelated segment with 10 coefficients

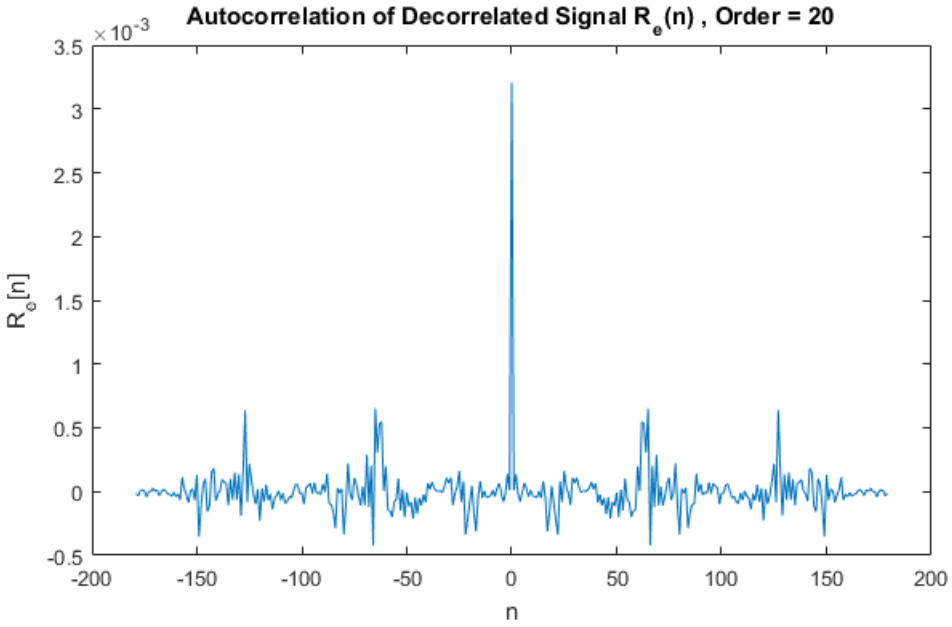

Figure 18: Autocorrelation function of decorrelated segment with 20 coefficients

4.2.3 Comparison of Power Spectral Density(PSD)

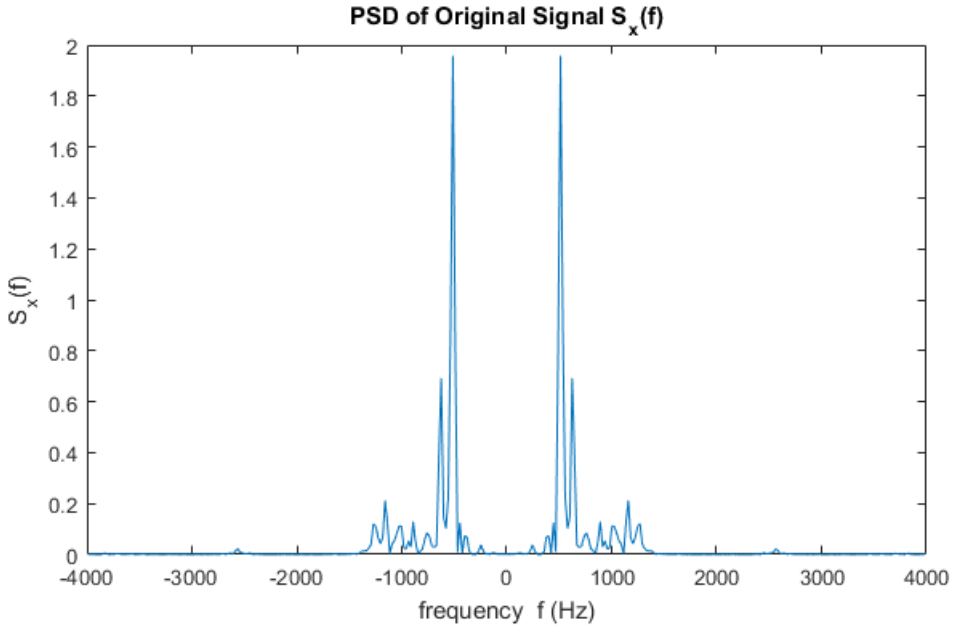

Figure 19: PSD of the original segment

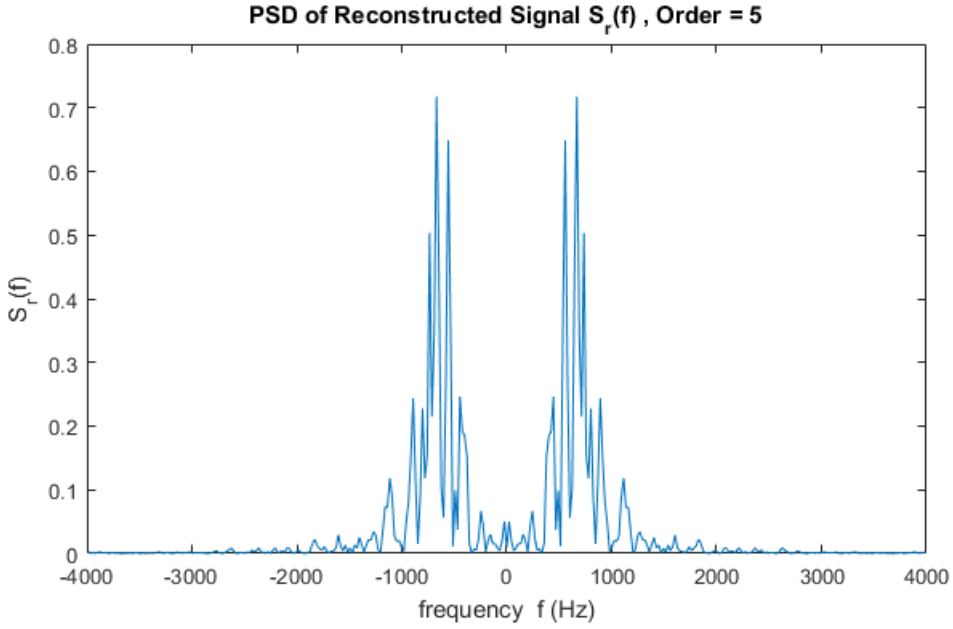

Figure 20: PSD of reconstructed segment with 5 coefficients

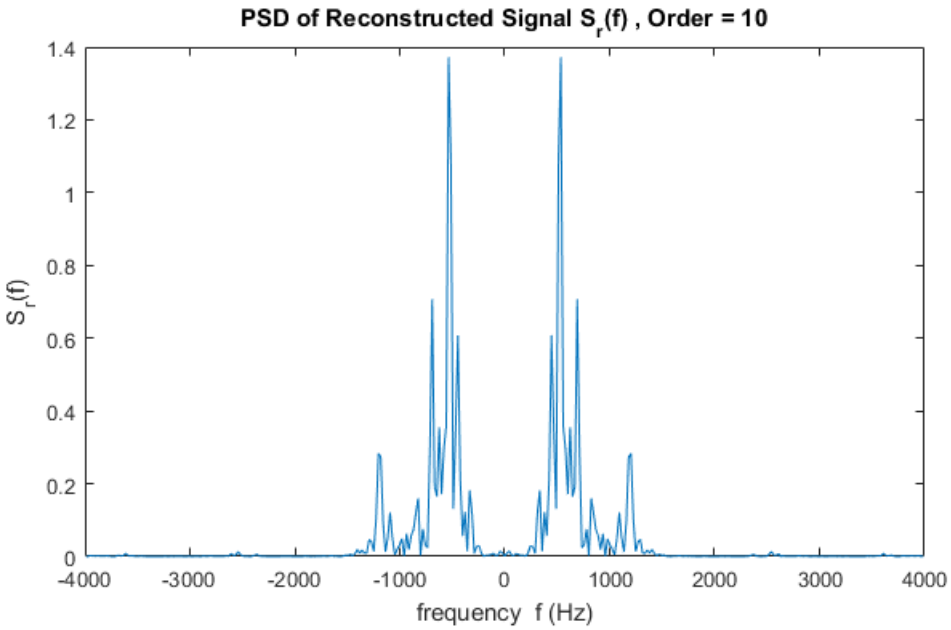

Figure 21: PSD of reconstructed segment with 10 coefficients

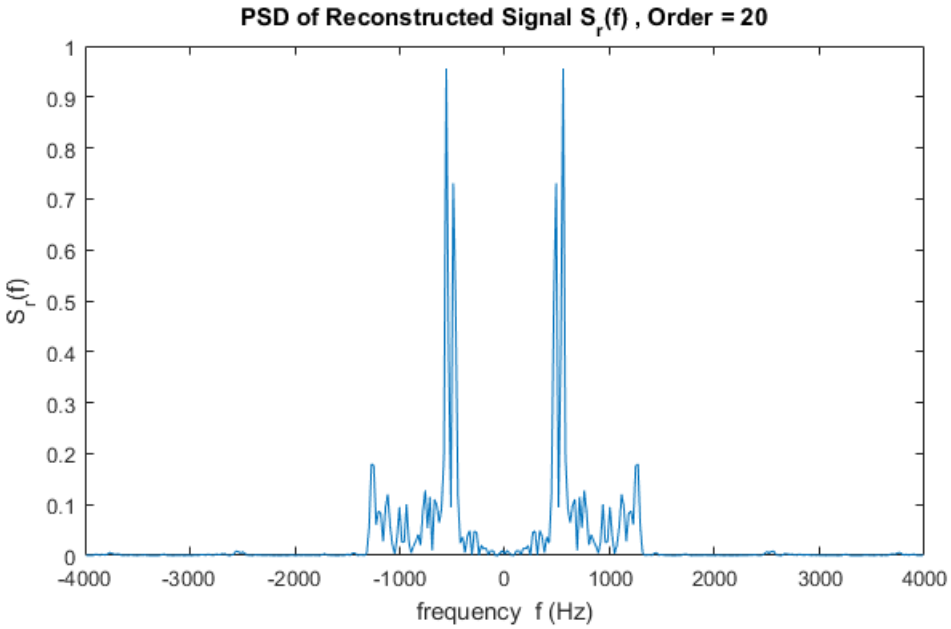

Figure 22: PSD of reconstructed segment with 20 coefficients

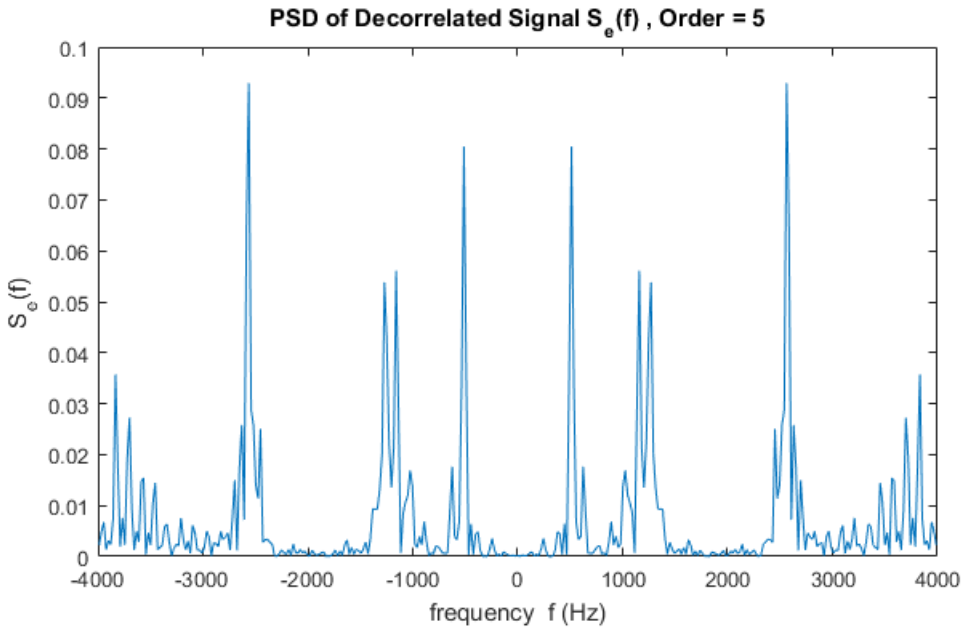

Figure 23: PSD of decorrelated segment with 5 coefficients

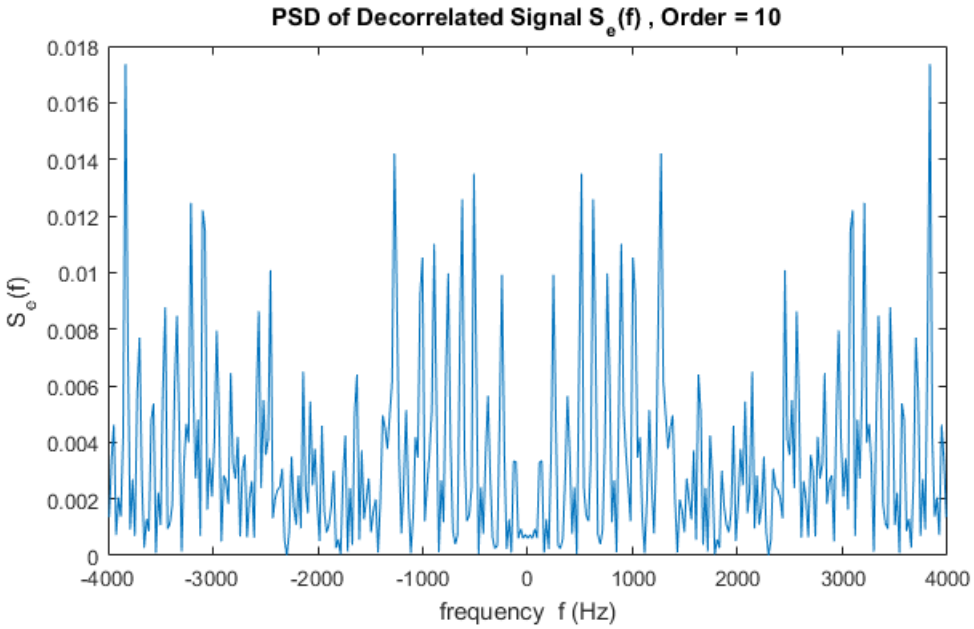

Figure 24: PSD of decorrelated segment with 10 coefficients

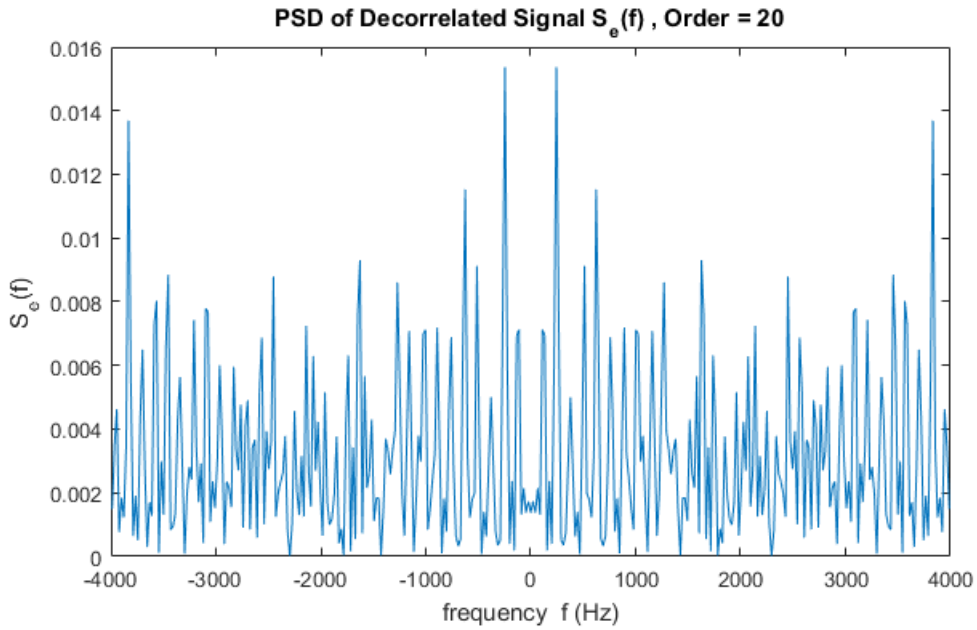

Figure 25: PSD of decorrelated segment with 20 coefficients

4.2.4 Implementation in C++

Matlab octave, being a high level programming language makes the implementation of any algorithms very easier, since it does not demand the deep understanding of the data structures. Hence it is difficult to optimise the computations which makes the real time implementation of an algorithm next to impossible. Here comes the relevance of relatively lower level programming languages like C/C++.

As done in the matlab implementation, a WAV file of speech signal sampled at 8 kHz is read into an array. Every WAV file contains a header of size 44 bytes at the beginning, which contains the information about the audio such as sampling rate, byte rate, number of channels (whether mono or sterio), total length of the audio etc. A data structure is realized containing specific fields to read these informations. After this, the sample values are stored as short integer of size 2 bytes varying from -32768 to 32767. To read these sample, a dynamic array of short integers is used which is varying in size according to the length of input audio.

The audio sample datas are divided into segments of length 22.5 millisecond. These segments are temporarily stored in an array and analysis are done in a loop using Levinson's Recursion algorithm. The order of prediction filter is taken as an input from the user and the filter coefficients are computed using lattice structure as explained in the theory. The prediction error filter is also implemented by lattice form realization. The filter

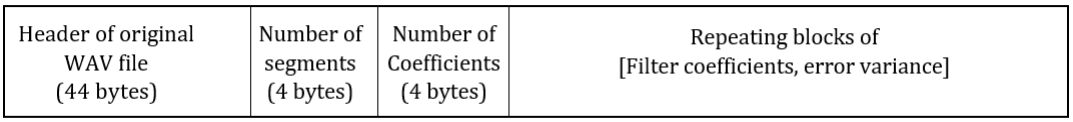

Figure 26: Structure of the binary file where the filter coefficients are written

coefficients are written to a binary file along with the error variance. The parameters of Linear Predictive Analysis (Number of filter coefficients and number of segments) are also written to this along with the WAV header of original file. The output of the prediction error filter (decorrelated signal) is written to another WAV file.

Another programme realizes the synthesis filter to reconstruct the audio. The filter coefficients and error variances are read from the binary file. A random generator object is used for generating white gaussian noise which excites the synthesis filter to produce the reconstructed wave form. While observing the result, the reconstructed audio is found to have high frequency noises. This encourages to realize a lowpass filter to refine the reconstruction. Thus a simple averaging is done on the output of synthesis filter to give a better result.

5 Conclusion

Linear prediction technique is an efficient tool in modelling of digital signals. An auto-regressive system with finite number of poles has been modelled. The filter coefficients, impulse response, magnitude and phase spectrum of the estimated system are compared with the original system, which gives a highlighting result. The same technique is applied on speech signals which finds a satisfactory result in modelling the speech production system and thereby generating speech waveforms from noise. The analysis is done using both autocorrelation and Levinson's recursion algorithm. Since the former goes for matrix inverse calculation, which is computationally complex, the algorithm fails in real-time applications. The Levinson's Recursion algorithm is computationally simpler and meets the requirements for real-time applications. The coefficient computation and prediction error filter are done in matlab using lattice form realization. The autocorrelation and power-spectral ldensity functions of original, reconstructed and decorrelated signals are studied. The speech coding part is implemented in C++ using fundamental data structures.

6 References

[1]Sophocles J. Orfanidis. "Linear Prediction" in *Optimum Signal Processing,* 2nd Edition. Rutgers School of Engineering: Collier Macmillan, 1988, pp. 147-160.
[2]T. F Quatieri. *Discrete-Time Speech Signal Processing.* Delhi: Pearson Education, 2006, pp. 595-665.

7 Appendix

7.1 Matlab code: Modelling of Auto-regressive System

```
close all;
clear all;
%% setting the poles and zeros
d = 0.5;
z = zeros(1,6);
poles = [cos(4*pi/5)+j*sin(4*pi/5),cos(2*pi/3)+j*sin(2*pi/3)]
poles = [poles,cos(pi/4)+j*sin(pi/4)];
poles = d*[poles, conj(poles)];
%%frequency response: evaluating H(z) along the unit circle in Z-plane.
f = [0:0.01:2*pi];
len = size(f,2);
gain = ones(1,len);
for k = 1:len;
    fcomp = cos(f(k))+j*sin(f(k));
    for kk = 1:6;
        gain(k) = gain(k)*(fcomp-z(kk))/(fcomp-poles(kk));
    end;
end;
%%plotting magnitude and phase spectrum
figure,subplot(2,1,1), plot(f,abs(gain));
title('Magnitude Response');
xlabel('Frequency');
ylabel('Magnitude');
subplot(2,1,2),plot(f,angle(gain));
title('Phase Response');
xlabel('Frequency');
ylabel('Phase');
%% Finding the filter coefficients using convolution technique
polynomial = [ones(5,1), -1*transpose(poles(1:5))];
res = [1,poles(3)];
for i = 1:2;
    res = conv(polynomial(i,:),res);
end;
res1 = [1,poles(6)];
for i = 4:5;
    res1 = conv(polynomial(i,:),res1);
end;
res = conv(res,res1);
%%exciting the system with a gaussian noise
len = 500;
```

```
x = normrnd(0,0.5,1,len);
y = zeros(1,len+5);
y(6) = x(1);
for k = 7:len;
    y(k) = x(k-5);
    for kk = 1:6;
        y(k) = y(k)-real(res(1+kk))*y(k-kk);
    end
end
figure,subplot(2,1,1),plot(x); %plotting the input and output signals
title('White Noise');
xlabel('Time');
ylabel('Magnitude');
subplot(2,1,2),plot(y(6:len+5));
title('Output signal');
xlabel('Time');
ylabel('Magnitude');
%% Linear predictive analysis on the output signal
y(1:5) = '';
cor = xcorr(y);
cor = cor./len;
corMat = zeros(7);
for k = 1:7
    corMat(k,:) = cor(len+1-k : len-k+7);
end
corMat = inv(corMat);
eps = zeros(7,1);
eps(1) = 1/corMat(1);
coef= corMat*eps;
%% Finding the impulse response of modelled system
len = 30;
ImpEst = zeros(1,len+5);
ImpEst(6) = 1;
for k = 7:len
    for kk = 1:6
        ImpEst(k) = ImpEst(k)-real(coef(1+kk))*ImpEst(k-kk);
    end
end
%% Finding the impulse response of actual system
Imp = zeros(1,len+5);
Imp(6) = 1;
for k = 7:len
    for kk = 1:6
        Imp(k) = Imp(k)-real(res(1+kk))*Imp(k-kk);
```

```
    end
end
%%plotting impulse responses for comparison.
figure,subplot(2,1,1), plot(ImpEst(6:len+5));
title('Impulse response of estimated system');
xlabel('Time');
ylabel('h[n]');
subplot(2,1,2),plot(Imp(6:len+5));
title('Impulse response of actual systeml');
xlabel('Time');
ylabel('h[n]');
%% frequency response of the estimated system
[r,p,k]=residuez([1 0 0 0 0 0 0],transpose(coef));
len = size(f,2)
gain1 = ones(1,len);
for k = 1:len
    fcomp = cos(f(k))+j*sin(f(k));
    for kk = 1:6
        gain1(k) = gain1(k)*(fcomp-z(kk))/(fcomp-p(kk));
    end
end
gain = gain/max(abs(gain));
gain1= gain1/max(abs(gain1));
%%Plotting the frequency responses of estimated and actual system
figure,subplot(2,1,2),plot(f,abs(gain));
title('Magnitude response of actual system');
xlabel('frequency'),ylabel('Magnitude');
subplot(2,1,1),plot(f,abs(gain1));
title('Magnitude response of estimated system');
xlabel('frequency'),ylabel('Magnitude');
figure,subplot(2,1,2),plot(f,angle(gain));
title('Phase response of actual system');
xlabel('frequency'),ylabel('Phase');
subplot(2,1,1),plot(f,angle(gain1));
title('Phase response of estimated system');
xlabel('frequency'),ylabel('Phase');
```

7.2 Matlab Code: Speech Coding - Method 1

```
clear all;
[a,fs] = audioread('aud.wav');
a = transpose(a);
a(fs*5:size(a,2)) = '';
n = round(.03*fs);
l = size(a,2);
l = l - mod(l,n);
a(l+1:size(a,2)) = '';
imax = l/n;
order = input('Enter Order:');
corMat= zeros(order);
coef = zeros(imax,order);
epsMin  = zeros(imax,1);
eps = zeros(order,1);
for i = 1:imax
    cor = xcorr(a(((i-1)*n+1):i*n));
    cor = cor/n;
    for j = 1:order
        corMat(j,:) = cor((n-j+1):n-j+order);
    end
    corMat = inv(corMat);
    epsMin(i) = 1/corMat(1,1);
    eps(1,1) = -epsMin(i);
    coef(i,:) = transpose(corMat*eps);
end;
synt = zeros(1,1);
short = zeros(1,n+order-1);
for i = 1:imax
    imp = normrnd(0,sqrt(epsMin(i,1)),1,n);
    for j = 1:n
        short(order-1+j) = 0;
        for k = 1:order-1
            short(order-1+j) = short(order-1+j) +  short(order-1+j-k)*coe
        end
        short(order-1+j) = short(order-1+j) + imp(j);
    end
    synt = [synt,short(order:n+order-1)];
end
input('Press Enter to play original audio');
soundsc(a,fs);
input('Press Enter to play reconstructed audio');
soundsc(synt/max(synt),fs);
```

7.3 Matlab Code: Speech Coding - Method 2

```
clear all;
close all;
[a,fs] = audioread('aud.wav');
a = transpose(a);
a(fs*5:size(a,2)) = '';
n = round(.0225*fs);
l = size(a,2);
l = l - mod(l,n);
a(l+1:size(a,2)) = '';
imax = l/n;
order = input('Enter Order:');
corMat= zeros(order);
coef = zeros(imax,order+1);
epsMin  = zeros(imax,1);
eps = zeros(order,1);
e = zeros(imax,n+order);
scale = n;
for i = 1:imax
    temp= a(((i-1)*n+1):i*n);
    gap = xcorr(temp)./scale;
    ap  = [1];
    for p = 0:order-1
        gamma = gap(n+p+1)/gap(n);
        r_gap = gap(2*n-1:-1:1);
        r_gap = [zeros(1,p+1), r_gap];
        r_gap(2*n:2*n+p) = '';
        gap = gap - gamma*r_gap;
        ap = [ap;0];
        ap = ap - gamma*ap(p+2:-1:1);
    end
    coef(i,:) = -transpose(ap);
    epsMin(i) = gap(n);
    e(i,:) = conv2(temp,transpose(ap));
end;
e(:,n+1:n+order) = '';
error = [0];
for i = 1:imax
    error = [error e(i,:)];
end
input('Press Enter to play original audio');
soundsc(a,fs);
input('Press Enter to play decorrelated audio');
```

```
soundsc(error,fs);
synt = zeros(1,1);
order = order+1;
short = zeros(1,n+order-1);
for i = 1:imax
    imp = normrnd(0,sqrt(epsMin(i,1)),1,n);
    for j = 1:n
        short(order-1+j) = 0;
        for k = 1:order-1
            short(order-1+j) = short(order-1+j) +  short(order-1+j-k)*coe
        end
        short(order-1+j) = short(order-1+j) + imp(j);
    end
    synt = [synt,short(order:n+order-1)];
end
synt = synt/max(synt);
corOriginal = xcorr(temp)/size(temp,2);
corError = xcorr(e(imax,:))/size(e,2);
corRecon = xcorr(short(order:n+order-1))/n;
figure, plot([-(size(temp,2)-1):1:(size(temp,2)-1)],corOriginal);
title('Autocorrelation of Original Signal R_x(n)');
xlabel('n');
ylabel('R_x[n]');
figure, plot([-(size(e,2)-1):1:(size(e,2)-1)],corError);
title('Autocorrelation of Decorrelated Signal R_e(n)');
xlabel('n');
ylabel('R_e[n]');
figure, plot([-n+1:1:n-1],corRecon);
title('Autocorrelation of Reconstructed Signal R_r(n)');
xlabel('n');
ylabel('R_r[n]');
psdOriginal = fftshift(abs(fft(corOriginal)));
psdError = fftshift(abs(fft(corError)));
psdRecon = fftshift(abs(fft(corRecon)));
freq = (fs/359)*[-179:1:179];
figure,plot(freq,psdOriginal);
title('PSD of Original Signal S_x(f)');
xlabel('frequency  f (Hz)');
ylabel('S_x(f)');
figure,plot(freq,psdError);
title('PSD of Decorrelated Signal S_e(f)');
xlabel('frequency  f (Hz)');
ylabel('S_e(f)');
figure,plot(freq,psdRecon);
```

```
title('PSD of Reconstructed Signal S_r(f)');
xlabel('frequency  f (Hz)');
ylabel('S_r(f)');
input('Press Enter to play reconstructed audio');
soundsc(synt,fs);
```

7.4 C++ Code: Speech Coding - Analysis

```
#include<iostream>
#include<fstream>
#include<vector>
#include<cmath>
using namespace std;

struct header   {
    char chunk_id[4];
    int chunk_size;
    char format[4];
    char subchunk1_id[4];
    int subchunk1_size;
    short int audio_format;
    short int num_channels;
    int sample_rate;
    int byte_rate;
    short int block_align;
    short int bits_per_sample;
    char subchunk2_id[4];
    int subchunk2_size;
};

void getCor(short* data, float* result,int n)   {
        /*
        The function assumes the input process is stationary.
        data    => input array
        result  => correlation matrix(output array)
        n       => length of input array
        */
        for(int i = 0; i < n; i++)      {
                result[i] = 0;
                for(int j = 0; j <= i; j++)
                        result[i] += data[j]*data[n-1-i+j];
                result[i] /= n;
                result[2*n-2-i] = result[i];
        }
}

int main()      {
        header meta;
        ifstream inFile;
        int n=0;
```

```
vector<short> data;
inFile.open("aud.wav");
if (!inFile.is_open()) {
        cout << "Unable to open the file"<<endl;
        exit(1);
}
inFile.read((char*)&meta, sizeof(meta));
if(meta.num_channels==2)        {
        short buffer[2];
        while(!inFile.eof())    {
                inFile.read((char*)buffer, sizeof(buffer));
                data.push_back((buffer[0]+buffer[1])/2);
                n++;
        }
}
else    {
        short buffer[2];
        while(!inFile.eof())    {
                inFile.read((char*)buffer, sizeof(buffer));
                data.push_back(buffer[0]);
                data.push_back(buffer[1]);
        }
}
inFile.close();
ofstream outFile, coeff;
outFile.open("decorrelated.wav");
coeff.open("coeff.bin");
coeff.write((char*)&meta, sizeof(meta));
outFile.write((char*)&meta, sizeof(meta));
int order,imax;
n = 180;
float gamma;
float coefficient;
short temp[180];
short r_temp[180];
short buffer[180];
short sample;
int g=0;
int w1=0, w2;
cout<<"enter order";
cin>>order;
float* r_gap= new float[359];
float* gap = new float[359];
float ap[40] = {0};
```

```
float r_ap[40] = {0};
imax=data.size()/180;
coeff.write((char*)&imax,sizeof(int));
coeff.write((char*)&order,sizeof(int));
cout<<order<<endl;
//Entering main loop
for(int i=0;i<imax;i++) {
        for(int l=0;l<n;l++)    {
                temp[l]=data[w1];
                r_temp[l] = data[w1];
                w1++;
        }
        getCor(temp,gap,180);
        for(int j = 0; j< order+2; j++) {
                ap[j] = 0;
                r_ap[j] = 0;
        }
        r_ap[1] = 1;
        ap[0] = 1;
        for(int p=0;p<order;p++)        {
                gamma=-gap[180+p]/gap[179];
                w2 = p+1;
                //updating gap function
                for(int j = 2*n-2; j > p; j--)
                        r_gap[w2++] = gap[j];
                for(int j = 0; j<p+1; j++)
                        r_gap[j] = 0;
                for(int y=0;y<359;y++)
                        gap[y] +=gamma*r_gap[y];
                //
                //lattice for filter coefficient
                for(int j = 0; j<p+2; j++)
                        ap[j] += gamma*r_ap[j];
                for(int j = 0; j<p+3; j++)
                        r_ap[j] = ap[p+2-j];
                //lattice form realization of
                //prediction error filter
                for(int j = 179; j > 0; j--)
                        r_temp[j] = r_temp[j-1];
                        // multiplying e_(n) by Z^(-1)
                r_temp[0] = 0;
                // adding a zero at the left
                for(int j = 0; j < 180; j++)
                        buffer[j] = temp[j];
```

```
                        // storing e(n) temporarily
                        for(int j = 0; j < 180; j++)
                                temp[j] += gamma*r_temp[j];
                        //updating e(n) in lattice
                        for(int j = 0; j < 180; j++)
                                r_temp[j] += gamma*buffer[j];
                        //updatig e_(n) in lattice
                }
                ap[0] = sqrt(gap[179]);
                for(int j = 0; j < order+1; j++)        {
                        coefficient = ap[j];
                        coeff.write((char*)&coefficient, 4);
                }
                for(int j = 0; j < 180; j++)    {
                        sample = temp[j];
                        outFile.write((char*)&sample,2);
                }
        }
        cout<<"Till here"<<endl;
        outFile.close();
        coeff.close();
        for(int i = 0; i<order+1; i++)
                cout<<ap[i]<<endl;
        return 0;
}
```

7.5 C++ Code: Speech Coding - Synthesis

```
#include<iostream>
#include<fstream>
#include<vector>
#include<random>

using namespace std;

struct header   {
    char chunk_id[4];
    int chunk_size;
    char format[4];
    char subchunk1_id[4];
    int subchunk1_size;
    short int audio_format;
    short int num_channels;
    int sample_rate;
    sampling rate.
    int byte_rate;
    short int block_align;
    short int bits_per_sample;
    char subchunk2_id[4];
    int subchunk2_size;
};

int main()  {
    vector<short> out;
    header meta;
    ifstream inFile;
    ofstream outFile;
    inFile.open("coeff.bin");
    outFile.open("recons.wav");
    inFile.read((char*)&meta,sizeof(meta));
    outFile.write((char*)&meta,sizeof(meta));
    int imax, order;
    inFile.read((char*)&imax,sizeof(int));
    cout<<imax<<endl<<order<<endl;
    inFile.read((char*)&order,sizeof(int));
    cout<<imax<<endl<<order<<endl;
    float coeff[40] = {0};
    short noise[180];
    short recons[180],sample;
    float sd;
```

```
default_random_engine generator;
normal_distribution<double> distribution(0,1);
for(int i = 0; i<imax; i++) {
    inFile.read((char*)&sd, sizeof(float));
    distribution = normal_distribution<double>(0,sd);
    for(int j = 0; j<180; j++)
        recons[j] = distribution(generator);
    for(int j = 0; j<order; j++)    {
        inFile.read((char*)&coeff[j],sizeof(float));
        cout<<coeff[j]<<" ";
    }
    cout<<endl;
    for(int j = 0; j < order+1; j++)
        for(int k = 0; k<j; k++)
            recons[j] -= coeff[k]*recons[j-k-1];
    for(int j = order+1; j<180; j++)
        for(int k = 0; k<order; k++)
            recons[j] -= coeff[k]*recons[j-k-1];
    for(int j = 0; j< 180; j++) {
        sample = recons[j];
        out.push_back(sample);
        outFile.write((char*)&sample, sizeof(short));
    }
}
inFile.close();
outFile.close();
ofstream out1;
out1.open("reconstructed_filtered.wav");
out1.write((char*)&meta,sizeof(meta));
cout<<"Enter the order of averaging filter :";
cin>>order;
float sample1;
for(int i = 0; i< order; i++)   {
    sample1 = 0;
    for(int j = 0; j<=i; j++)
        sample1 += out[j];
    sample = sample1/order;
    out1.write((char*)&sample,sizeof(short));
}
for(int i = order; i<out.size(); i++) {
    sample1 = 0;
    for(int j = 0; j<order; j++)
        sample1 += out[i-j];
    sample = sample1/order;
```

```
            out1.write((char*)&sample,sizeof(short));
        }
        out1.close();
        return 0;
    }
```

7.6 Matlab Commands Used

7.7 C++ Libraries Used

- xcorr(A): Computation of autocorrelation
- conv(A,B): Convolution of two arrays
- residuez(b,a): To solve higher degree numerator and denominator polynomials
- Other elementary commands

7.8 C++ Libraries Used

- $< iostream >$: for basic input/output services
- $< fstream >$: for file processing
- $< vector >$: for arrays of dynamic type
- $< random >$: for random variables