CHAPTER: 3 PROPOSED SYSTEM

The proposed model recognizes the speech using the wiener and the volterra series. The Wiener filter is used for the noise reduction, i.e. to improve the speech quality. The enhanced speech is then recognized by the volterra series. While recognition output of the trained signal is compared with the output of the test signal (speech). The closest match is given as the output. The description of wiener and volterra and proposed model is given below:

3.1 VOLTERRA AND WIENER
The most effective way to represent the complex i.e. nonlinear system is volterra and Wiener series. These representations are more effective in the speech processing as well as the signal processing. These representations are extended version of the linear system representation. In this, the convolution of the input signal and its impulse response is taken or in other words, the input is multiplied by various nonlinear coefficients of the input signal.

3.2 NONLINEAR SYSTEMS REPRESENTATION
Impulse response is used to represent the linear systems. While for the nonlinear systems, there is no such representation that covers all non-linear systems. Volterra proposed a representation by extending the convolution method. In this representation a series of non-linear coefficients are used to represent the complex systems. These operators are known as the volterra operators.

In the Volterra as well as in the Wiener series same operators i.e. volterra operators are used. Volterra operator is basically is the sum of multiplied component of input with itself. The degree of the operator is dependent on the number product terms. For example, the second-order Volterra is the sum of all pair wise multiplication of the input signal at different times. While the First-order Volterra operator is the weighted sum of the input signal at different times.

The basic difference between the representation of the volterra and Wiener series are the degree of the volterra operator. In the volterra series the operator must be of same degree. It means any
operator cannot be estimated independently. While in the wiener series different order of volterra operator can be used i.e. mixed degree of operator can be used. So the operator can be estimated independently. One more difference is that the volterra calculates the approximation to the original system while the wiener does the same for mean square.

3.2.1 Volterra: Basic Definition and Properties

A Volterra of order \( p \) with \( m \) components, \( X=\{x[n]\} \), is defined as:

\[
x[n] = \begin{cases}
a_{1,0} + \sum_{i=1}^{p} a_{1,i} x[n-i] + \varepsilon_1[n] & \text{w.p. } W_1(x[n-1]) \\
a_{2,0} + \sum_{i=1}^{p} a_{2,i} x[n-i] + \varepsilon_2[n] & \text{w.p. } W_2(x[n-1]) \\
\vdots \\
a_{m,0} + \sum_{i=1}^{p} a_{m,i} x[n-i] + \varepsilon_m[n] & \text{w.p. } W_m(x[n-1])
\end{cases}
\]

Where \( \varepsilon_i \) is a zero-mean Gaussian random process with a variance of \( \sigma^2 \), “w.p.” denotes “with probability” and the gating weights, \( W_i \) sum to 1 and these weights are defined by the following equation:

\[
W_i(x) = \frac{w_i + g_i x}{\sum_{j=1}^{m} w_j + g_j x}
\]

The linear prediction coefficients, \( \{a_i\} \), represent the dynamic model, where \( a_{i,0} \) are the component means, while \( \{w_i, g_i\} \) are called gating coefficients. It is apparent that an \( m \)-mixture Volterra process is the weighted sum of \( m \) Gaussian autoregressive processes with the time-dependent weights depending on previous data and the gating coefficients.

One convenient way of viewing this model is as a process in which each data samples at any one point in time is generated from one of the component AR mixture processes chosen randomly according to its weight \( W_i \). An overview of a 2-component Volterra model is illustrated in the following figure.
3.3 MODELING NONLINEARITIES USING VOLterra

One property of Volterra that is of particular relevance, here is the ability of Volterra to model nonlinearity in time series [91, 92] though the individual component AR processes are linear. The probabilistic mixing of these AR processes constitutes a nonlinear model. Even when the mixture weights are fixed, the model reduces to MAR, which is still nonlinear. It is to be noted here that even though GMM also employs probabilistic mixing of components, because of the static nature of the components - i.e., each component is a single value of a random variable and not a random process – it cannot model nonlinear dynamics in the data. The addition of a gating system layer for weight generation increases the flexibility of the model even further, allowing us to model distributions as a function of past data even better.

Even simple MAR dynamics can lead to chaotic patterns in data. For example, we can generate fractals out of seemingly trivial MAR models. The famous Sierpinski triangle fractal can be generated out of the following MAR model using only three components and even using only fixed weights:

\[
X[n] = \begin{cases} 
X[n-1] + A_1 & \text{w.p. } W_1 \\
X[n-1] + A_2 & \text{w.p. } W_2 \\
X[n-1] + A_2 & \text{w.p. } W_3 
\end{cases}
\]
Where $X$ is the trajectory of the data points on the 2-D plane, and $A_i$'s are three fixed points on the plane. An example Sierpinski triangle generated from such a model is shown in the figure above. One indication of a chaotic signal is the bandwidth of the power-spectrum. Most natural signals that are not chaotic exhibits a low-bandwidth in their power spectrum, while chaotic signals have a large bandwidth power spectrum resembling that of a stochastic system. For example, 50,000 samples were generated according to the following Volterra model:

$$X[n] = \begin{cases} 
0.5X[n-1] + 0.0 & \text{w.p. 0.33} \\
0.2X[n-1] - 1.0 & \text{w.p. 0.33} \\
0.3X[n-1] + 1.0 & \text{w.p. 0.33} 
\end{cases}$$

Eq. 3.4

A snapshot of the signal and the associated power spectrum are depicted in the figure below.

![Figure 23: Example of a Volterra generated signal and its power spectrum; the broad-band nature of power spectrum indicates that the signal is chaotic](image)

From the power spectrum, it can be seen that Volterra signals are broadband signals, and hence possibly chaotic.

### 3.4 VOLTERRA SERIES
The voltage series can be represented in the time domain by using non-linear operator say $T$. The relation between the input $x$ and the output $y$ is denoted below.

$$y(t) = T x(t)$$

The method seems to be constant with the time, and it is continuous in nature. In other words, same input gives the duplicate output at any interval of time. However, if there is any alteration in the input, then corresponding output gets changed. Conventionally we can say the system is limited the $T$ to be a linear operator $H_1$ in such a way that the system can be defined by the convolution of input with its impulse response, i.e. $h(1)$.

$$y(t) = H_1 x(t) = \int_{\mathbb{R}} h^{(1)}(\tau) x(t-\tau) \, d\tau$$

The above series can be extended for the nonlinear systems by introducing nonlinear integral operators.

$$y(t) = h^{(0)} + \int_{\mathbb{R}} h^{(1)}(\tau_1) x(t-\tau_1) \, d\tau_1 :$$

$$+ \int_{\mathbb{R}^2} h^{(2)}(\tau_1, \tau_2) x(t-\tau_1) x(t-\tau_2) \, d\tau_1 d\tau_2 :$$

$$+ \int_{\mathbb{R}^3} h^{(3)}(\tau_1, \tau_2, \tau_3) x(t-\tau_1) x(t-\tau_2) x(t-\tau_3) \, d\tau_1 d\tau_2 d\tau_3 :$$

$$+ \ldots$$

Overall Volterra series can define the system as shown

$$y(t) = H_0 x(t) + H_1 x(t) + H_2 x(t) + \ldots + H_n x(t) + \ldots$$

In which $H_0 x(t) = h^{(0)}$ is a constant and for $n = 1, 2, \ldots$

$$H_n x(t) = \int_{\mathbb{R}^n} h^{(n)}(\tau_1, \ldots, \tau_n) x(t-\tau_1) \ldots x(t-\tau_n) \, d\tau_1 \ldots d\tau_n$$
Here, \( H_n \) is the Volterra operator of degree \( n \).

The time interval can be decided by the extracting the system features. The time interval can be finite or infinite. In the volterra series the present output is affected by the past values i.e. the system is dependent on the previous output as well as on the current input. The volterra is basically the extended version of the Taylor series. The past dependent Taylor series is the volterra series. In other words, the Taylor series generates the output by using the present input only while the Volterra series generates the output by using the current input as well as the previous output. The Volterra operator is not unique for any particular input. There are various operators who can lead to the same output \([94]\) \([95]\).

The convergence of the input speech is necessary due to the power series feature of the volterra. The complex system is mandatory to be approximated as the input and output may belong to different class. If the input is limited to square-integral functions on any particular interval, then approximation by volterra series is enough \([96]\) \([97]\). Sometimes these results seem to be general but to limitation applied on the input, the results are sufficient. The main application of the volterra series is the analysis of the nonlinear, i.e. complex systems by using differential equations. In the signal processing or speech recognition, it uses the separate and finite amount of data. The sectorial data can be produced from any multi-dimensional input or by using different images or time series. A discrete system is simply mentioned by a function that maps these vectors to some real number, not by an operator as in the continuous time case. The discretized Volterra operator \([98]\) is defined as the function

\[
H_n x = \sum_{i_1=1}^{m} \ldots \sum_{i_n=1}^{m} h_{i_1 \ldots i_n}^{(n)} x_{i_1} \ldots x_{i_n}.
\]

Here the volterra operator is the combination of the nth order polynomial components of input.

### 3.5 Wiener Series

The expansion of any function using the Taylor series may lead to the deviation from the initial point. The convergence of the volterra series is similar to the convergence of the Taylor series. In this convergence, the error and derivation must tend to zero or a very small value. The
convergence can be restricted to the mean square sense if the function used is the orthogonal function, and the range is grater as compared to the range of Taylor's series.

The output of the different functional of volterra series can be correlated as they are not orthogonal. However, the orthogonality is required, and it can be applied by using Gram-Schmidt orthogonal but it is only on average of all input. So a probability value is assigned to each input signals in other words a stochastic process over which the Volterra functional are orthogonalized in terms of the correlation between their outputs over all possible inputs. The resulting functionals are sums of Volterra functionals of different order or non-homogeneous Volterra functionals. As the process is selected by the Wiener for the series of non homogeneous Volterra functionals so the process is known as Wiener process [99]. In this probability is assigned to the every input possible by the given input function [100]. The process of realizations for the Wiener is similar to the sinusoidal test inputs in linear system theory.

The orthogonalization converts the non homogeneous Volterra functionals to the Wiener functional that can be denoted by $G_n$ [101]. The corresponding input-specific decomposition of the system operator $T$

$$y(t) = G_0 x(t) + G_1 x(t) + G_2 x(t) + \ldots + G_n x(t) + \ldots$$

Into a series of mutually uncorrelated operators is called a **Wiener series** expansion. The orger of the Wiener operators $G_n$ is the n as it is linear combinations of n Volterra operators having maximum degree n. The second-degree Wiener operator (for instance)

$$G_2 x(t) = \int h_2(\tau_1, \tau_2) x(t - \tau_1) x(t - \tau_2) \, d\tau_1 d\tau_2 - \int h_2(\tau_1, \tau_1) \, d\tau_1$$

The zero-order and the second-order Volterra operator can be seen. The third-degree Wiener operator is:

$$G_3 [x(t)] = \int_{\mathbb{R}^3} k_3(\tau_1, \tau_2, \tau_3) x(t - \tau_1) x(t - \tau_2) x(t - \tau_3) \, d\tau_1 d\tau_2 d\tau_3 :$$
An odd-degree Wiener operator contains all lower odd-order Volterra operators and an even-degree Wiener operator contains all lower even-order Volterra operators. The Wiener kernel can be calculated by the recursive formula [102] given below:

\[-3A \int_{\mathbb{R}^2} k_3(\tau_1, \tau_2, \tau_2) x(t - \tau_1) \, d\tau_1 \, d\tau_2\]

The zero and first-degree Wiener functionals do not contain derived Wiener kernels and are given by

\[G_0[x(t)] = k^{(0)} \quad \text{and} \quad G_1[x(t)] = \int_{\mathbb{R}} k^{(1)}(\tau_1) x(t - \tau_1) \, d\tau_1.\]
The discrete analogue to the Wiener series is typically orthogonalized with respect to Gaussian input $x \sim N(0,A)$ since this is the only practical setting where the popular cross correlation method can be applied. The properties of continuous Wiener series operators described above carry over to the discrete case. Any square-summable function with Gaussian input can be approximated in the mean square sense by a finite, discrete Wiener series [105][106].

### 3.6 ESTIMATION OF VOLterra AND WIENER SERIES

To recognize any system only input and the output signals are given, no other information of the system is known. The different representation (volterra and wiener) have to minimize the error between the actual output and output from the model. One has to solve a set of integral equation to minimize the MSE i.e. mean square error. Wherever the continuous series is needed then volterra can’t be used. In such situations the only possibility is to use the wiener series as in this series all the operators are independent.

If the system is linear then the response of the system can be evaluated easily by some basic signals like sinusoids. But to evaluate the non-linear system, the response to each input is needed. We can say for the non-linear system the response to each class input signal is different. All possible input signals is infeasible since testing, they resort to inputs that are realizations of a random process. This process is feasible to calculate any continuous input signal in the least squares sense. If the input time series is sufficiently long, then obtained system representation should generalize to be previously unseen continuous input.

The wiener operator are most popular as they can be evaluated by using cross-correlation method of Lee and Schetzen [116]. If one uses Gaussian white noise with standard deviation $A$ instead of the Wiener process as input, the leading Wiener kernel can be calculated as the cross-correlations

$$k^{(0)} = \overline{y(t)}$$

$$k^{(1)}(\sigma_1) = \frac{1}{A} \overline{y(t)x(t - \sigma_1)}$$
\[ k(2)(\sigma_1, \sigma_2) = \frac{1}{2A^2} \overline{y(t)x(t-\sigma_1)x(t-\sigma_2)} \]

\[ k(3)(\sigma_1, \sigma_2, \sigma_3) = \frac{1}{3!A^3} \overline{y(t)x(t-\sigma_1)x(t-\sigma_2)x(t-\sigma_3)} \]

\[ k(n)(\sigma_1, \ldots, \sigma_n) = \frac{1}{n!A^n} \overline{y(t)x(t-\sigma_1)\ldots x(t-\sigma_n)} \]

Where the bar indicates the average over time. However, the Wiener operators are orthogonal by definition although the single Wiener kernels can be estimated independently. The lower-order Wiener kernels of \( G_n \) can be derived from the leading kernel by applying again formula.

The cross correlation method suffers from severe problems that limit its applicability to relatively small Wiener kernels in a low-noise scenario.

In practice, the cross-correlations have to be estimated at a finite resolution. The number of coefficients \( h(n) \) in Equation increases with \( mn \) for an \( m \)-dimensional input signal and an \( n \)-th order Volterra kernel. Since, the number of coefficients that actually have to be estimated by cross-correlation is less. Since the products in Eq. remain the same when two different indices are permuted and the associated coefficients should be equal. The required number of measurements is \( (n+m-1)!/(n!(m-1)!) \) [95] as a consequence. Nonetheless, the resulting numbers are huge for higher-order Wiener kernels. For instance, the 5th-order Wiener kernel operating on 16 x 16 sized image patches contains roughly 1012 coefficients out of which 1010 would have to be measured individually by cross-correlation. This procedure is not feasible for higher-dimensional input signals or highly nonlinear kernels with larger memory.

The estimation of cross-correlations requires large sample sizes. Typically, they need several tens of thousands input-output pairs before a sufficient convergence is reached. The variance of the estimator \( \overline{y(t)x(t-\sigma_1)\ldots x(t-\sigma_n)} \) increases with increasing values of the \( \sigma_i \) [99] such that only operators with relatively small memory can be reliably estimated.
The estimation via cross-correlation works only if the input is Gaussian noise with zero mean, never for general types of input. In physical experiments, deviations from ideal white noise and the resulting estimation errors cannot be avoided. On the other hand, specific inputs may have a very low probability of being generated by white noise. However, the approximation is only computed in the mean square sense. The system response may be drastically different from the model predictions as compared to these inputs.

With the advent of readily available computing power, the cross-correlation method has been replaced in most cases by linear regression, which does not suffer from the cumbersome estimation of cross-correlations and the restriction to Gaussian noise input. They directly estimate the discrete Volterra series by treating the monomials as basis functions

\[ \varphi_{i_1 \ldots i_n} = x_{i_1} \ldots x_{i_n} \] and the components of Volterra kernel \( h_{i_1 \ldots i_n}^{(n)} \) as expansion coefficients

\[ f(x) = \sum_{n=0}^{p} H_n x = \sum_{n=0}^{p} \sum_{i_1=1}^{m} \ldots \sum_{i_n=1}^{m} h_{i_1 \ldots i_n}^{(n)} x_{i_1} \ldots x_{i_n} = \sum_{n=0}^{p} \sum_{i_1=1}^{m} \ldots \sum_{i_n=1}^{m} h_{i_1 \ldots i_n}^{(n)} \varphi_{i_1 \ldots i_n}. \]

Instead of assuming an infinite amount of data, expansion coefficients are found by minimizing the mean squared error over a finite dataset consisting of \( N \) input-output pairs (\( x_j, y_j \)) \[107],[95]\]

\[ \frac{1}{N} \sum_{j=1}^{N} (f(x_j) - y_j)^2. \]

This minimization problem is solved by applying the standard numerical methods for least square’s problems such as pseudo inverses or singular value decomposition, without the need of estimating cross correlations. Moreover, the input signal class is no more restricted to Gaussian noise, but it can be chosen freely, e.g., from the ‘natural’ input ensemble of the system. As long as possible the input is known to the experimenter, there is no requirement for controlling the input as in the classical system identification setting. Note that the obtained Volterra models will approximate the Wiener series only for sufficiently large datasets of Gaussian white noise. Korenberg et al. [117] have shown that the linear regression framework leads to Wiener models that are orders of magnitude more accurate than those obtained from the cross-correlation method. The solution of this regression problem requires the inversion of an \( M \times M \) matrix where
M is the number of monomials in the expansion [94]. This is again prohibitive for high-dimensional data and higher orders of nonlinearity since M scales like \( mn \).

This dimensionality problem can be overcome by using kernel regression instead of standard linear regression [107]. It can be shown that all finite discrete Volterra series of degree \( p \) those are solutions of a linear regression problem can be generated by an expansion in polynomial kernel functions \((1+x^T j x)^p\) with suitable weights \( \alpha_j \), the implicit Volterra series

\[
f(x) = \sum_{n=0}^{p} H_n x = \sum_{j=1}^{N} \alpha_j (1 + x^T j x)^p.
\]

This means that, instead of \( mn \) coefficients \( h(n) i_1...i_n \), only \( N \) coefficients \( \alpha_j \) of the implicit Volterra series have to be estimated. This is advantageous if the number of the monomials exceeds the number of available data points which is typically the case for high-dimensional inputs and higher orders of non-linearity. Implicit Volterra series can be converted into a standard discrete Volterra series expansion [108]. If the \( \alpha_j \) are computed from Gaussian white noise input by minimizing the mean squared error, finally resulting Volterra series is called an implicit Wiener series from which also an expansion into a standard discrete Wiener series can be computed [108]. The kernel function \((1+x^T j x)^p\) is not the only one that is capable of representing Volterra series. There are several alternatives some of them generate also infinite Volterra series.

3.7 COMPARISON OF VOLTERRA TO OTHER MODELS

It is easy to find parallels between the Volterra and GMM models. In particular, Volterra can be viewed as a generalization of GMM that models each component as a sum of the output of an autoregressive filter with a specified mean, and with mixture of weights determined by a gating system similar to a mixture of experts. It should be noted that with the component orders and \( g_i \) set to zero, Volterra, reduces to the familiar GMM. This similarity between the two makes it straightforward to replace GMM with Volterra for speaker recognition.

In a GMM, the distribution remains invariant to the past samples due to the static nature of the model. For Volterra, the conditional distribution given past data varies with time. This model is
capable of modeling both the conditional means and variances. Thus, Volterra can model time series that evolve nonlinearly. Some other properties of Volterra, including a mathematically rigorous proof of the ability of Volterra’s to be arbitrarily closely model stochastic processes are derived in [108]\textbf{Reference source not found.}. Note that in the original formulation, both the gate and prediction orders were constrained to be equal. In this work, we restrict our use of Volterra order to one to avoid difficulties during parameter estimation.

One of the earliest applications of autoregressive HMMs (AR-HMMs) considered an autoregressive filter to model state observations in a 5-state HMM for speaker verification [109]\textbf{Reference source not found.}. A more recent investigation of AR-HMMs [110] used a switching autoregressive process to capture signal correlations during state transitions. Results on speech recognition showed that at best their model was only comparable to an LPC-based HMM using a GMM observation model. Another model considered speech features as a GMM white noise process filtered through an autoregressive signal for speaker identification [111].

### 3.8 VOLTERRA PARAMETER ESTIMATION

Similar to the well-known training procedure for GMM, maximum likelihood estimates for Volterra prediction and variance parameters can be calculated using the Expectation-Maximization (EM) algorithm. Given the order, \( p \), the parameter set for each of the \( m \) components of an MAR model consists of \( p + 1 \) predictor coefficients (including the mean), the error variance, and mixing weight:

\[
\theta_i = \{a_i,0,a_i,1,\ldots,a_i,p,\sigma_i,w_i,g_i\}_{l=1,\ldots,m}.
\]

To estimate these parameters, we first need an initial guess for these parameters and then we iterate with EM to successively refine the estimates. An initialization strategy that we found to work reasonably well was to first train a GMM with the same number of mixtures and then set each component of the Volterra to have the same mean, variance, and weight as the GMM
model. We initialize the predictor coefficients and the data-dependency gating coefficients, \(\{A_i\}\) of Volterra to zero.

These initial parameters can be then refined recursively using an E-step [108]:

\[
\gamma_l[n] = \frac{W_l p_l(x[n]|\theta)}{\sum_{k=1}^{m} W_k p_k(x[n]|\theta)}
\]

Where

\[
p_l(x[n]|\theta) \propto \frac{1}{\sigma_l} e^{-\frac{1}{2\sigma_l^2} (x[n] - a_{l,0} - \sum_{i=1}^{m} a_{l,i} x[n-i])^2}
\]

is the probability a sample was generated from component \(l\) at time instant \(n\). The corresponding M-step is given by:

\[
\hat{A}_l = R_l^{-1} r_l
\]

where

\[
R_l = \sum_{n=p+1}^{N} \gamma_l[n] X_{n-1} X_{n-1}^T
\]

\[
r_l = \sum_{n=p+1}^{N} \gamma_l[n] X_{n-1} x[n]
\]
Refer to comments on estimation of predictor coefficients and variances for Volterra and MAR in [119][92] for further details. However, a complication arises with respect to the estimation of gating coefficients for Volterra. There is no closed-form solution for these, and hence a Newton gradient-ascent approach must be used:

\[
X_{n-1} = \begin{bmatrix} 1 \\
x[n-1] \\
x[n-2] \\
\vdots \\
x[n-p] \end{bmatrix}.
\]

\[
\hat{w}_l = w_l + \beta \frac{\Delta Q}{\Delta w_l}
\]

\[
\hat{g}_l = g_l + \beta \frac{\Delta Q}{\Delta g_l}
\]

Where \( Q \) denotes the log-likelihood of the Volterra model for the training data. \( \beta \) and \( \Delta \) are design parameters to be chosen empirically. The expression for computing \( Q \) is:

\[
Q(\theta) = \sum_{n=1}^{N} \sum_{l=1}^{m} \gamma_l[n] \log(W_l[n]) + \sum_{n=1}^{N} \sum_{l=1}^{m} \gamma_l[n] \log(p_l(x[n]|\theta))
\]

Due to this complication in the updates for the gate coefficients, the training procedure outlined above is not in the realm of strict EM algorithm but falls into a class of algorithms called as generalized EM algorithms (GEM) [112]. For both EM and GEM algorithms, the E-step is similar. However, while an EM algorithm actually maximizes the expectation during each M-step, a GEM algorithm only guarantees that parameters that increase the model likelihood for the data is increased but does not guarantee that his is maximized at each M-step. This could mean that a GEM algorithm could take more number of iterations for training than an EM algorithm for the same or a comparable problem.
Drawing parallels with the choice of an adaptation factor in adaptive filter theory, we can envisage that quick and smooth convergence of the GEM algorithm can be achieved by starting with a relatively high value for $\beta$ and then reducing this value with successive iterations. In our experiments, we found that fixing $\Delta = 0.01$ and running 10 iterations each with $\beta = 0.9$, $\beta = 0.5$, and $\beta = 0.2$ in succession provided a reasonably smooth and quick convergence. One example to illustrate the convergence of the EM algorithm for Volterra with algorithmic values set as mentioned above is shown in Figure below:

![Figure 24: EM Convergence as a function of iterations on a synthetically synthesized signal](image)

This is for an 8 mixture Volterra training on a synthetic signal generated from a Volterra model of speaker 4516 of NIST-2001 development database [113]. From this, we can see that we achieve reasonably quick convergence with these algorithmic parameter values. However, such convergence is not guaranteed in general and this poses a problem to the application of this model for real-life signals.
Fortunately, we can do better than guessing an appropriate value for beta. We can use the secant method for root-finding and maximization [113]. In general, to find the maxima using Newton method, the iteration is:

$$\hat{x} = x + \frac{f'(x)}{f''(x)}$$

In the secant method, the double derivative in the denominator is estimated numerically using the secant at the point. Thus, we estimate the scaling factor $\beta$ as the inverse of double derivative of the log-likelihood w.r.t. the gate parameters:

$$\beta = 1/ \frac{\Delta^2 Q}{\Delta^2 w_l}$$

During implementation, this scheme amounts to finding for each gate coefficient $w_l$, the value of $Q$ at three different points, $Q(w_l)$, $Q(w_l + \Delta)$, $Q(w_l - \Delta)$, and then using the following update equation:

$$\hat{w}_l = w_l + \frac{Q(w_l + \Delta) - Q(w_l - \Delta)}{Q(w_l + \Delta) + Q(w_l - \Delta) - 2Q(w_l)}$$

Similarly, the update equation for gate coefficients $g_l$ is:

$$\hat{g}_l = g_l + \frac{Q(g_l + \Delta) - Q(g_l - \Delta)}{Q(g_l + \Delta) + Q(g_l - \Delta) - 2Q(g_l)}$$

Using this method, we obtain convergence curve shown in the following figure for the same data from speaker 4516 of NIST-2001 database [113] used for the previous method. We find that this method is more reliable and quick - three GEM iterations were sufficient. We use this method in future experiments.
Figure 25: Performance of (Generalized) EM using secant method, as a function of iterations for a 8-mixture Volterra model of speaker data from “4516” of NIST-2001 database.

3.9 Wiener Filter

Wiener filter can be used for signal enhancement to remove the effect of linear distortions such as the de-blurring of distorted or unfocused images or equalization of the distortion of a telecommunication channel, or noise reduction.

Wiener filter represented by the filter’s coefficient vector \( w \). Filter takes as the input a signal \( y(m) \), usually a distorted version of a desired signal \( x(m) \), and produces an output signal \( \hat{x}(m) \), where \( \hat{x}(m) \) is the least mean square error estimate of the desired or target signal \( x(m) \). The filter input–output relation is given by

\[
\hat{x}(m) = \sum_{k=0}^{p-1} w_k y(m-k) = w^T y
\]
Where \( m \) is the discrete-time index, vector \( y^T = [y(m), y(m-1), \ldots, y(m-P-1)] \) is the filter input signal, \( \hat{x}(m) \) is the filter output and the parameter vector \( w^T = [w_0, w_1, \ldots, w_{P-1}] \) is the Wiener filter coefficient vector.

In the frequency domain, Wiener filter output \( \hat{X}(f) \) is the product of the input signal \( Y(f) \) and the filter frequency response \( W(f) \):

\[
\hat{X}(f) = Y(f)W(f)
\]

The estimation error signal \( E(f) \) is defined as the difference between the desired signal \( X(f) \) and the filter output \( \hat{X}(f) \) as

\[
E(f) = X(f) - \hat{X}(f) = X(f) - Y(f)W(f)
\]

Consider a signal \( x(m) \) observed in a broadband additive noise \( n(m) \), and modelled as

\[
y(m) = x(m) + n(m)
\]

Assuming that the signal and the noise are uncorrelated i.e, \( r_{xn}(m) = 0 \), it follows that the autocorrelation matrix of the noisy signal is the sum of the autocorrelation matrix of the signal \( x(m) \) and the noise \( n(m) \):

\[
R_{yy} = R_{xx} + R_{nn}
\]

and we can also write

\[
r_{xy} = r_{xx}
\]

Where \( R_{yy}, R_{xx}, R_{nn} \) are the autocorrelation matrices of the noisy signal, the noise-free signal and the noise respectively, and \( r_{xy} \) is the cross-correlation vector of the noisy signal and the noise-free signal.

Substitution of Equations, yields

\[
w = (R_{xx} + R_{nn})^{-1}r_{xx}
\]
Above equation is the optimal linear filter for the removal of additive noise. The following study of the frequency response of the Wiener filter provides useful insight into the operation of the Wiener filter.

In the frequency domain, the noisy signal \( Y(f) \) is given by

\[
Y(f) = X(f) + N(f)
\]

Where \( X(f) \) and \( N(f) \) are the signal and noise spectra. A signal is observed in additive random noise, frequency Wiener filter is obtained as

\[
W(f) = \frac{P_{XX}(f)}{P_{XX}(f) + P_{NN}(f)}
\]  \hspace{1cm} (1)

Where \( P_{XX}(f) \) and \( P_{NN}(f) \) are the signal and noise power spectra. While dividing the numerator and the denominator of the (1) Equation by the noise power spectra \( P_{NN}(f) \) and substituting the variable \( SNR(f) = \frac{P_{XX}(f)}{P_{NN}(f)} \) yields

\[
W(f) = \frac{SNR(f)}{SNR(f)+1}
\]  \hspace{1cm} (2)

Where SNR is a signal-to-noise ratio measure. Note that in Equation (2) the variable \( SNR(f) \) is expressed in terms of the power-spectral ratio and not in the more usual terms of log power ratio or dB. Therefore \( SNR(f) = 0 \) corresponds to zero signal content or \( 10 \log_{10}(0) = -\infty \) dB and \( SNR(f) = 1 \) corresponds to equal signal and noise power \( P_{XX}(f) = P_{NN}(f) \) or \( 10 \log_{10}(1) = 0 \) dB and \( SNR(f) = 0.5 \) corresponds to \( 10 \log_{10}(0.5) = -3 \) dB.

**3.10 PROPOSED MODEL**

\[\text{Input Speech} \rightarrow \text{Pre-emphasis} \rightarrow \text{Harmonic Regeneration Weiner Filter} \rightarrow \text{Analog to Digital Converter} \]

\[ \text{O/P} \rightarrow \text{Non-linear Volterra Predictor} \rightarrow \text{Feature Extractor} \rightarrow \text{Hamming Window} \]

*Figure 26: Block Diagram*
3.10.1 Pre-emphasis
In the Pre-emphasis phase, the energy of the input speech is boosting in the high frequencies. The glottal pulse causes a spectral tilt i.e. more energy at lower frequencies than higher frequencies. The boosting of high-frequency energy gives more information to the acoustic model. It also enhances the recognition performance.

3.10.2 Harmonic Regeneration Wiener Filter
Speech enhancement using the Weiner filter results in some harmonics, which considered as noise only components. To solve the problem harmonic structure of the speech is taken into the account. Here, the output signal of Wiener algorithm is further processed to create an artificial signal where the missing harmonics have been automatically regenerated. Then this artificial signal is used to compute a suppression gain that tries to preserve all the harmonics of the clean speech signal. The detail block diagram shows the processing of harmonic regenerator wiener filter.

3.10.3 Principle of harmonic regeneration
A simple and efficient way to restore speech harmonics consists of applying a nonlinear function NL to the time speech segment obtained from a Wiener filter. Then the artificially restored segment is obtained by

\[ x_{\text{harm}_n}(n) = NL(\hat{x}_n(n)) \quad (1) \]

The restored harmonics of \( x_{\text{harm}_n}(n) \) are created at the same positions as the clean speech ones. It is very interesting and important characteristic of the method is ensured because of nonlinearity in the time domain.

In the enhanced speech segment obtained with wiener filter have some harmonics completely suppressed. The artificially restored segment can be obtained by equation (1) where the nonlinearity has restored the harmonics. However, the harmonic amplitudes of the artificial signal are biased compared to clean speech.
Figure 27: Brief block diagram
3.10.4 A/D Conversion
The Analog speech signal is converted into a digital form by A/D converter. In the training phase, a .wav file is created with following parameters:

- Sampling frequency : 16 kHz
- Type of coding : Pulse Code Modulation
- Precision : 16 bit stereophonic
- Recording rate : 62 kbps.

Then this .wav file is converted into a vector matrix by using `wavread` command in Matlab, during training phase of the system.

3.10.5 Windowing
The speech signal is blocked into frames of N samples with adjacent frames being separated by M samples. If M= (1/3) N samples, then the first illustrated frame consists of the first N speech samples. Second frame begins M samples after first sample of the first frame and overlaps it by N-M samples. Same as, the third frame begins 2M samples after the first frame and overlaps it by N-2M samples. The process continues until all speech is accounted for one or more frames. It is easy to see that M<=N. For a 20 ms period the no. of samples will be 16000*20*10e (-3) = 320 samples, where 16000 is the sampling frequency and 20*10(-3) sec i.e. 20 ms is frame duration.

The next step in the processing is to a window each individual frame to minimize the signal discontinuities at the beginning and end of each frame. The Hamming window is used which has the form

\[
    w(n) = \begin{cases} 
    0.54 - 0.46 \cos \left( \frac{2\pi n}{N-1} \right) & 0 \leq n \leq N-1 \\ 
    0 & \text{elsewhere} 
    \end{cases}
\]
3.10.6 Feature Extraction

Feature extraction means converts the speech waveform to some of the type of parametric representation. The parametric representation is then used for further analysis and processing. The features of speech signal are amplitude of the signal, energy, intensity, velocity, acceleration, fundamental frequency and vibration rate, etc. Feature extraction is the process of obtaining different features such as pitch, power and vocal tract configuration from the speech signals. Here the speech signal is represented in the Volterra parametric form.

3.10.7 Non-linear Volterra Predictor

A nonlinear predictor based on Volterra filters estimates a current signal value by a linear combination of past signal values, and additionally by linear combinations of products of past signal values. Hence, the predictor is nonlinear in the signal values yet linear in the filter coefficients. As a consequence, adaptation algorithms valid in the linear case can be extended to Volterra filters. Without loss of generality, the system will be treated with the first and second-order kernels only. Then, the predicted signal is given by

\[ \hat{x}(n) = \sum_{k=1}^{p} h_1(k) \cdot x(n-k) + \sum_{i=1}^{p} \sum_{j=1}^{p} h_2(i,j) \cdot x(n-i) \cdot x(n-j) \]

where \( \hat{x}(n) \) is the estimation of \( x(n) \) and \( p \) is the prediction order. The coefficients \( h_1(k) \) and \( h_2(i,j) \) represent linear and nonlinear components, respectively. The symmetry of the coefficients is assumed, so \( h_2(i,j) = h_2(j,i) \). In this case, the overall number of coefficients for the second order Volterra predictor equals

\[ n_c = p + \frac{p \cdot (p + 1)}{2} \]

The short-term predictor eliminates the correlation between nearby samples. This is well known that the prediction order must be high enough to include at least one pitch period, in order to model a voiced signal adequately. However, this is not acceptable for most practical implementations due to large delay and increased complexity. The standard solution to this
problem in linear prediction is the use of a model with two predictors, short-term and long-term, connected in cascade. A long-term predictor in such a realization targets correlation between samples one or multiple pitch periods apart. This solution is used in a number of speech coders.

Since the number of coefficients is not a critical issue in Volterra long-term prediction, the third-order predictor can be used as well in cascade with a short-term linear predictor. In this case, the number of coefficients is increased only by two compared to linear LTP. The corresponding predicted sample equals

\[ \hat{e}_x(n) = h_1 \cdot e_x(n - T) + h_2 \cdot e^2_x(n - T) + h_3 \cdot e^3_x(n - T) \]

Minimizing the sum of squared errors the following LTP coefficients are obtained

\[
h_1 = \frac{q_2 \cdot q_4 \cdot q_7 - q_3 \cdot q_4 \cdot q_5 + q_5^2 \cdot q_6 - q_3 \cdot q_7 \cdot q_8 - q_2 \cdot q_5 \cdot q_6 + q_5^2 \cdot q_8}{q_3^2 + q_2^2 \cdot q_7 + q_1 \cdot q_5^2 - q_1 \cdot q_3 \cdot q_7^2 \cdot q_2 \cdot q_3 \cdot q_5}
\]

\[
h_2 = \frac{q_1 \cdot q_5 \cdot q_6 - q_3 \cdot q_5 \cdot q_8 - q_2 \cdot q_3 \cdot q_6 + q_2 \cdot q_7 \cdot q_8 - q_1 \cdot q_4 \cdot q_7 + q_3^2 \cdot q_4}{q_3^2 + q_2^2 \cdot q_7 + q_1 \cdot q_5^2 - q_1 \cdot q_3 \cdot q_7^2 \cdot q_2 \cdot q_3 \cdot q_5}
\]

\[
h_3 = \frac{q_1 \cdot q_4 \cdot q_5 - q_2 \cdot q_3 \cdot q_4 + q_5^2 \cdot q_6 - q_2 \cdot q_5 \cdot q_8 - q_1 \cdot q_3 \cdot q_6 + q_3^2 \cdot q_8}{q_3^2 + q_2^2 \cdot q_7 + q_1 \cdot q_5^2 - q_1 \cdot q_3 \cdot q_7^2 \cdot q_2 \cdot q_3 \cdot q_5}
\]

Where

\[
q_1 = \sum_{n} e^2_x(n - T)
\]

\[
q_2 = \sum_{n} e^3_x(n - T)
\]

\[
q_3 = \sum_{n} e^4_x(n - T)
\]
The speech recognition process is completed in two phases

I) Training
II) Testing

In the First phase, the proposed model is applied and the results, i.e. output of the model are stored. In the MATLAB, the storage is performed using the MAT files. Here, 50 words are stored for the training purpose. In the second phase, the output of testing signal is compared with the stored values. The name corresponding to value having same order and the least mean difference is given as the output.