Linear Prediction Algorithms

Mohit Garg (00D07015)
$http : //www.ee.iitb.ac.in/uma/\tilde{}mohit$
Indian Institute of Technology, Bombay,India

April 24, 2003

Abstract

Estimating a variable given some other variables, is one of the most commonly occuring problems in real life. More specifically, estimation or prediction is one of the most celebrated problems in time series analysis. This report is a MATLAB implementation of two algorithms for the same - Levinson-Durbin Algorithm and the Weighted Least Squares Error Algorithm.

Keywords: Linear Prediction, Levinson-Durbin, Least Squares

1 Introduction

One of the most celebrated problems in time series analysis is that of *prediction i.e.* given a series of sample values of a stationary discrete-time process (*e.g.* a signal), we need to *predict* its future samples. Specifically, given $x(n-1), x(n-2)....., x(n-M)$, we need to predict the value of $x(n)$.

In general, we may express this predicted value as a function of the given M past samples. *i*.e. i

$$\hat{x}(n|n-1, n-2, ..., n-M) = \Psi(x(n-1), x(n-2), ..., x(n-m)) \qquad (1)$$

Now, if this function Ψ is a linear function of the variables $x(n-1), x(n-2), ..., x(n-M)$, we say that the *prediction* is *linear*. We can visualize this in a $M - dimensional$ space spanned by $x(n-1), x(n-2), ..., x(n-M)$. Hence, we can write

$$\hat{x}(n|n-1, n-2, ..., n-M) = \sum_{k=1}^{M} a_k x(n-k) \tag{2}$$

where a_k are constant coefficients. Such a predictor can be realised by using a Tapped-Delay Line filter (Fig.1) .The prediction error is defined as

$$f_M(n) = x(n) - \hat{x}(n|n-1, n-2, ..., n-M) \tag{3}$$

Here, the subscript M in $f_M(n)$ denotes the *order* of the prediction. *i.e.* the number of past samples that are used to predict the next sample.

Hence, the problem of *Linear Prediction*, that we are interested in, reduces to determining these coefficients subject to some condition. Different algorithms and conditions on $a_k's$ have been proposed and are used. We will consider two common ones in this report: *Minimum Mean Square Error* and *Weighted Least Square Error*.

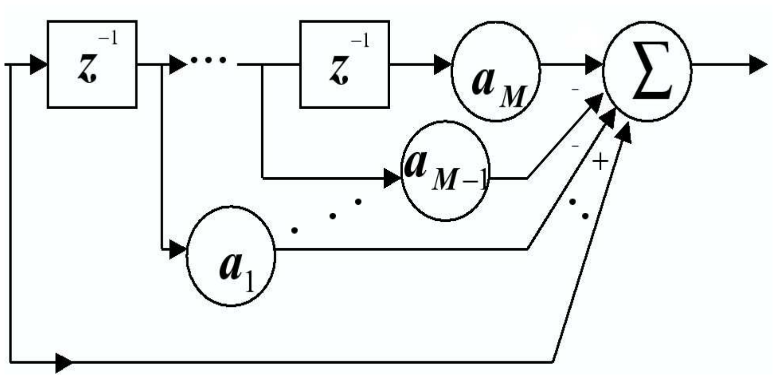


Figure 1: Tapped-Delay Line Realisation of the Linear Predictor(source [7])

2 Minimun Mean Square Estimate (MMSE)

Obviously, we would like to minimise the error in the predicted value and the actual value. But since, the sequence of variables is probabilistic, we will need to define some measure of equality or error. A commonly used measure for this is probability theory is the *RMS Error*, *i.e.*, Root Mean Square Error.[1]. RMS error is defined as

$$P_M = E(|f_M(n)|^2) \tag{4}$$

Minimising the prediction RMS error (P_M), we get the Weiner-Hopf equations (refer [5]) *i.e.*

$$\mathbf{Ra} = \mathbf{r} \tag{5}$$

where,

$$\mathbf{a} = \begin{bmatrix} a_1 & a_2 & a_3 & . & . & a_M \end{bmatrix}^{\mathbf{T}}$$

$$\mathbf{r} = \begin{bmatrix} R_{XX}(1) & R_{XX}(2) & R_{XX}(3) & . & . & R_{XX}(M) \end{bmatrix}^{\mathbf{T}}$$

$$\mathbf{R} = \begin{bmatrix} R_{XX}(0) & R_{XX}(-1) & R_{XX}(-2) & ... & R_{XX}(1-M) \\ R_{XX}(1) & R_{XX}(0) & R_{XX}(-1) & ... & R_{XX}(2-M) \\ R_{XX}(2) & R_{XX}(1) & R_{XX}(0) & ... & R_{XX}(3-M) \\ . & . & . & ... & . \\ . & . & . & ... & . \\ R_{XX}(M-1) & R_{XX}(M-2) & R_{XX}(M-3) & ... & R_{XX}(0) \end{bmatrix}$$

Here, $R_{XX}(k)$ denotes the autocorrelation function $(E(x(n)x(n-k)))$ of the sequence $x(n)$ for a lag k. Note that $R_{XX}(-k) = R_{XX}(k)$ for real signals(which we are interested in), since the process is assumed to be stationary. In order to solve for the coeff. a_k, we need to

- First, determine the autocorrelation function upto order M for the input process $x(n)$.
- Then, solve the above equations.

Determination of the autocorrelation function is addressed at the end of the section. Here we proceed assuming that the autocorrelation function upto order M is available.

[1]An estimate found by minimising the Root Mean Square Error is often referred to as the MMSE estimate

2.1 The Levinson-Durbin Recursion

Assuming we have already found out the autocorrelation functions for the input process $x(n)$, we need to solve the $M \times M$ system of equations to get the desired coefficients (a_k). One can use standard methods for solving linear equations, but the structure of the $\mathbf{R}$ matrix gives us some very unique advantages. The *Levinson-Durbin Recursion*[2], named in recognition of its use first by Levinson(1947) and then its independednt reformulation at a later date by Durbin (1960), is a direct recursive method for solving for the coefficients of the prediction filter. It makes particular use of the *Toeplitz structure*[3] of the matrix $\mathbf{R}$. A detailed derivation and proof of the method can be found in [2].

Here we outline the basic steps involved in the algorithm. The algorithm uses, the prediction filter coefficients of order $m-1$ to compute the coefficients of the filter of order m.
For $m = 1$ to M

1. Calculate *mth* order reflection coefficient $\Gamma_m = -\frac{\Delta_{m-1}}{P_{m-1}}$
2. Calculate the coefficients for the *mth* order prediction-error filter, given by
$$\hat{a}_{m,k} = \hat{a}_{m-1,k} + \Gamma_m \hat{a}^*_{m-1,m-k}, \qquad k = 0, 1, ...m$$
where,
$$\hat{a}_{M,k} = \begin{cases} 1 & k = 0 \\ -a_{M,k} & k = 1, 2, ..., M \end{cases}$$
3. Calculate the RMS error for the *mth* order filter as $P_m = P_{m-1}(1 - |\Gamma_m|^2)$
4. Calculate $\Delta_m = \mathbf{r}_m^{BT}\mathbf{a}_{m-1}$. Here $\mathbf{r}_m^{BT} = \begin{bmatrix} R_{XX}(m) & R_{XX}(m-1) & ... & R_{XX}(1) \end{bmatrix}$.

The algorithm is initialised by setting $\hat{\mathbf{a}}_0 = 1, P_0 = R_{XX}(0), \Delta_0 = R_{XX}(1)$.

2.2 Calculation of the Autocorrelation coefficients

Tha autocorrelation function of the input process may not be know apriori. Hence, we will need to estimate it based on the input process itself. In

[2]The Levinson-Durbin algorithm is also applicable when we donot know the autocorrelation function beforehand[2]

this implementation, I have replaced the expectations by the time averages (assuming ergodic process) and found out the autocorrelation function for lags $k = 1, 2, ..M$. *i.e.*

$$R_{XX}(k) = \frac{1}{N-k} \sum_{l=1}^{N} x(l)x^*(l-k)$$

This estimation of the autocorrelation function is not practical since it assumes the apriori knowledge of the entire process. There are other algorithms[3] for determining $R_{XX}(k)$, but have not been considered here. It is because of this practical problem, that we get motivated to go for the *Weighted Least Squares Error Algorithm* which does not suffer from this shortcoming and is adaptive in nature.

3 Weighted Least Squares Error (WLSE)

In the MMSE based prediction, we were minimising the RMS error in the predicted and actual values. In the WLSE Algorithm, we minimise another definition of the error. Here we take the weighted sum of the error and minimise it for a given set of weights. Note that we will be finding out a new set of filter coefficients at each instant ot timt $t = k$, and using those coefficients to predict the value at the next instant of time $t = k + 1$. In other words, we will be adaptively changing the coefficients in order to meet our minimum WLSE criterion. *i.e.*

$$\beta(\mathbf{a}) = \frac{1}{2} \sum_{i=1}^{k} \alpha_i [\mathbf{a}_M^T \mathbf{u}(i) - x(i)]^2 \tag{6}$$

where α_i are the weights and $\mathbf{u}(k)$ is the input to the filter at time $t = k$ *i.e.*

$$\mathbf{u}(k) = \begin{bmatrix} x(k-1) & x(k-2) & ... & x(k-M) \end{bmatrix}^T$$

Also,

$$\hat{x}(k) = \mathbf{a}_M^T(k-1)\mathbf{u}(k)$$

The weighting should de-emphasise the old data points since, in real life practical implementation of the algorithm, storage capacity will be limited

[3] *e.g.* Burg's Algorithm

and if we store all past data samples, we may run out of memory. Hence, we need to decrease the weights of the older samples and this can be acheived by selecting the weights in an appropriate way. We take $\alpha_k = \alpha^{k-1}$ where $\alpha < 1$. The variable is sometimes also referred to as the *forgetting factor*. The value of α depends on the nature of the input process. Usually $\alpha = 0.99$ is used and it acts as a trade-off between storing all past samples (after sufficient time, $\alpha^{k-1} << 1$ and hence, can be neglected altogether) and also allowing global optimisation to occur.
Due to the adaptive nature of the WLSE algorithm, the error between the predicted and actual values initially is large and starts falling as $t = k$ increases. The rate of convergence is relatively fast. A detailed mathematical treatment of the WLSE algorithm is given in [1]. Skipping the maths, we outline the algorithm below.

For $k = 2$ to ∞

1. Calculate the current predicted output $\hat{x}(k) = \mathbf{a}_M^T(k-1)\mathbf{u}(k)$

2. Update the coefficient vector

$$\mathbf{a}_M(k) = \mathbf{a}_M(k-1) + \frac{\mathbf{P}(k-1)\mathbf{u}(k)}{\alpha + \mathbf{u}^T(k)\mathbf{P}(k-1)\mathbf{u}(k)}[x(k) - \hat{x}(k)]$$

3. Update the $\mathbf{P}$ matrix

$$\mathbf{P}(k) = \frac{1}{\alpha}\left\{\mathbf{P}(k-1) - \frac{\mathbf{P}(k-1)\mathbf{u}(k)\mathbf{u}^T(k)\mathbf{P}(k-1)}{\alpha + \mathbf{u}^T(k)\mathbf{P}(k-1)\mathbf{u}(k)}\right\}$$

We start the algorithm with $\mathbf{a}_M = [1, 0, 0, ..., 0]^T$ and $\mathbf{P}(1) = \mathbf{I}$, the $M \times M$ Identity Matrix. Hence, we can adaptively estimate the next sample of the input process. This method can be used for a real time implementation of a Linear Predictor and does not require us to calculate the autocorrelation function of the input process.

4 Results

4.1 Sinusoidal Input

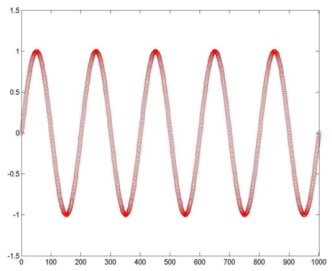


Figure 2: Sinusoidal Input to the Filters

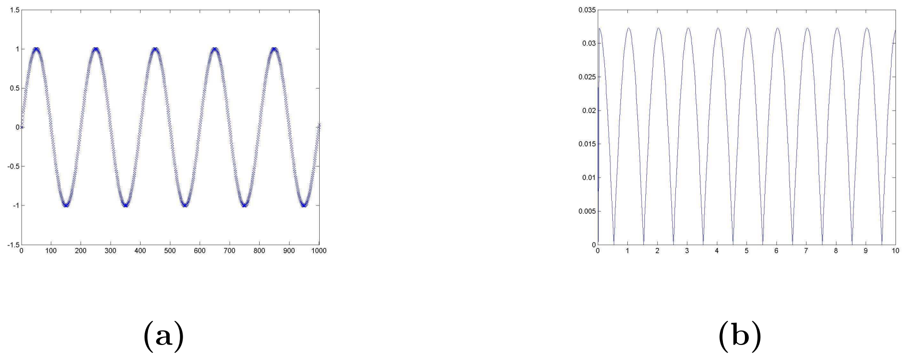
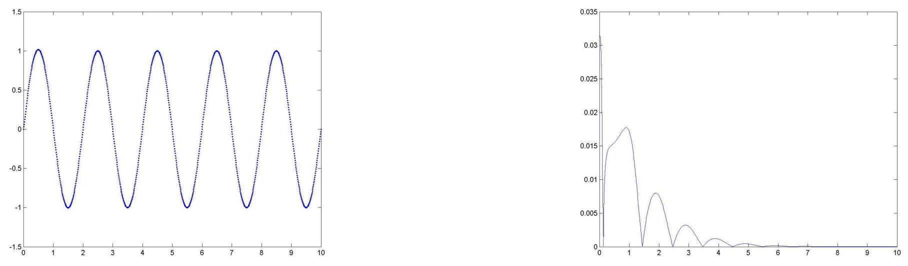


Figure 3: Output and Error for MMSE predictor



(a) **(b)**

Figure 4: Output and Error for WLSE predictor with $alpha = 0.99$

4.2 Image as input

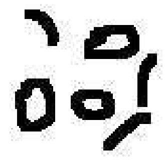


Figure 5: Image Input to the Filters

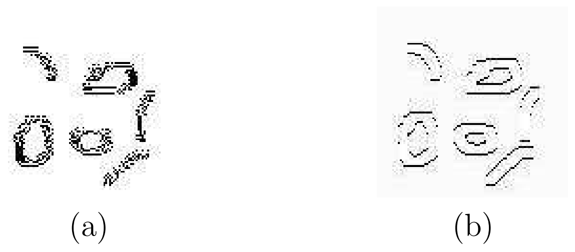


Figure 6: Output and Error for MMSE predictor

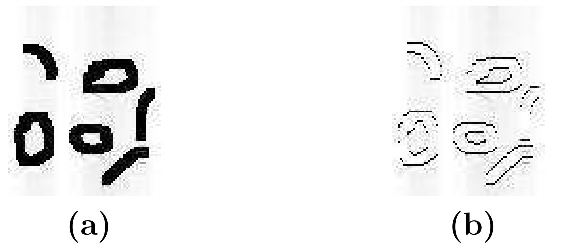
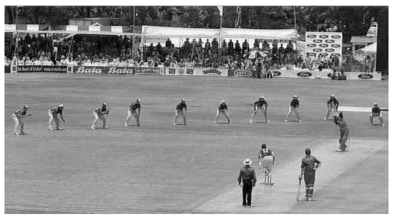


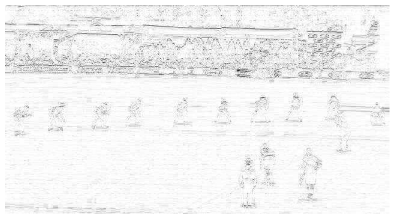
Figure 7: Output and Error for WLSE predictor with $alpha = 0.99$

4.3 Another Image

Figure 8: Image Input to the Filters

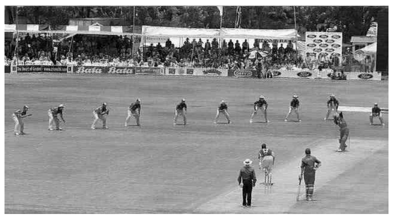


(a)

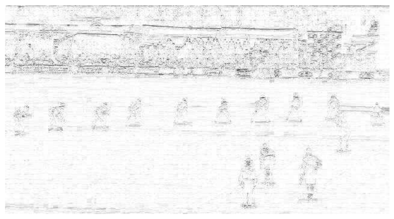


(b)

Figure 9: Output and Error for MMSE predictor



(a)



(b)

Figure 10: Output and Error for WLSE predictor with $alpha = 0.99$

5 Conclusion

The MMSE estimator and the WLSE estimator were simulated successfully.

The amount of computation effort required was reduced especially in the case of the MMSE estimator by using the Levinson-Durbin algorithm and hence avoiding relatively computationally expensive Gaussian Elimination.

The WLSE estimator is more applicabe to real life situations than the MMSE predictor (in the form studied) since it avoids calculation of the autocorrelation function.

We also see from the results that the predicted output closely matches with the input process especially in case of signals like images which we encounter in real life.

Linear Prediction is very useful in lots of applications especially signal processing. A whole class of coding schemes, especially for speech signals, called Linear Prediction Coding (LPC) exist. Linear Prediction is also used in Digial Communication systems *e.g.* Differential Pulse Code Modulation

(DPCM) which give a very good data compression and hence improve the multiplexing ability of the channel.

References

[1] Robert A. Monzingo, Thomas W. Miller, Introduction to Adaptive Arrays, John Wiley & Sons, 1980

[2] S. Haykin, Adaptive Filter Theory, Prentice Hall, New Jersey, 1986

[3] www.ulib.org/webRoot/Books/Numerical_Recipes/bookcpdf/c2-8.pdf

[4] P. Strobach, Linear Prediction Theory-A Mathematical Basis for Adaptive Systems, Springer-Verlag, 1990

[5] S. Haykin, Communication Systems, 4th Ed., John Wiley & Sons,2001

[6] http://www.ensc.sfu.ca/people/faculty/cavers/ENSC810/classnotes/classnotes810.htm

[7] http://hitchcock.dlt.asu.edu/media3/a_spanias/eee506-s02/PDF-506/EEE506-LECT09.pdf

Appendix

Here is the listing of the MATLAB code.

Code for the MMSE predictor using the Levinson-Durbin Algorithm
****

```
M=5; % order of the predictor
x=input('Enter the signal(row vector): ');

%for M=1:200
r=zeros(M+1,1); % Column matrix of the autocorrelation

%Calculating the autocorrelation functions upto order M
for k=0:M
    for l=1:length(x)
        if((l-k)>=1)
            r(k+1)=r(k+1)+(x(l)*conj(x(l-k)));
        end
    end
    r(k+1)=r(k+1)/(length(x)-k);
end

r0=r(1);
r=r(2:M+1,1); % remove r(0) from the matrix

% Now compute the coeff. (here we are using direct matlab inv. function)
%a=inv(R)*r % the coefficients
% Levinson-Durbin Recursion to get the coefficients
a=1; % a0 (start)
P=r0; % Power in error
D=r(1);
for m=1:M
    Gamma=-D/P; % The reflection coeff. for mth order filter
    a=[a;0] + Gamma.* conj(flipud([a;0])); % The coeff. for the mth order
        % filter
    P=P*(1-(abs(Gamma))^2); % Power of the error for mth filter
    if(m+1<=M)
        D=(flipud(r(1:m+1))')*a; % calculate Delta(D)(m)
    end
end
a=-a(2:M+1); % remove the first element of the Prediction-error filter
     % coeff. vector (since it is equal to 1 and we donot
     % need it here) also we need to put a negative sign
     % as the prediction-error-filter has a -ve sign for
             % prediction-filter coeff.
P

% Now we need to estimate the signal
s=zeros(size(x)); % estimate of x
s(1)=x(1);

for k=2:length(s)
    for l=1:M
        if(k-l>=1)
            s(k)=s(k)+a(l)*x(k-l);
        end
    end
end
%end

figure;
plot(1:length(x),x,'r-',1:length(s),s,'b-');
figure;
plot(1:length(s),abs(fft(s-x)))
```

Code for the WLSE adaptive-predictor
****

```
M=5; % order of the predictor
alpha=0.99; % forgetting factor

x=input('Enter the signal(row vector): ');

% initialise the coeff. vector
a=zeros(M,1);
a(1)=1;

% initialise the P matrix
P=eye(M);  % init. to idenity matrix

% initialise the output vector
s=zeros(size(x));
s(1)=x(1);

for k=2:length(x) % prediction starts at t=2;

    currin=zeros(M,1);
    % current input vector
    if(k-M>=1)
        currin=flipud(x(k-M:k-1)'); % converted to column vector
    else
        for z=1:M
            if(k-z>=1)
                currin(z)=x(k-z);
            end
        end
    end

    % calculate the predictor output at time t=k
    s(k)=a'*currin;

    % update coeff. for next iteration
    a=a+((P*currin)./(alpha+(currin')*P*currin))*(x(k)- s(k));
    P=1/(alpha).*(P-((P*currin*(currin')*P)./(alpha+(currin')*P*currin)));

end

figure;
plot(1:length(x),x,'r-',1:length(s),s,'b-');
figure;
plot(1:length(s),abs(fft(s-x)))
```