A Comparative Study of Time-Delay Estimation Techniques Using Microphone Arrays

School of Engineering Report No. 619

Yushi¹ Zhang and Waleed² H. Abdulla
Department of Electrical and Computer Engineering
The University of Auckland, Private Bag 92019,
Auckland, New Zealand

¹yzha104@ec.auckland.ac.nz  ²w.abdulla@auckland.ac.nz

2005
ABSTRACT

Time delay estimation (TDE) between signals received at two microphones has been proven to be a useful parameter for many applications. Speech enhancement, speaker localization, speech and speaker recognition and meeting activity detection are some examples of applications based on TDE. Five different time delay estimation methods are described in this paper and implemented using MATLAB. These methods are cross-correlation (CC), phase transform (PHAT), maximum likelihood estimator (ML), adaptive least mean square filter (LMS) and average square difference function (ASDF). Their simulation results are compared in terms of computation complexity, hardware implementation, precision, and accuracy. Since the performances of the TDE methods are considerably degraded by the signal-to-noise ratio (SNR) level, this factor has been taken as a prime factor in benchmarking the different methods.

Key words: TDE, CC, PHAT, ML, ASDF, LMS, SNR
I. INTRODUCTION

During the last forty years, the problem of estimating the time delay between signals received at two spatially separated microphones in the presence of noise has been considered for a variety of applications; such as in acoustics, radar communication, microphone array processing systems and speech recognition.

This physical problem in two dimensions is shown in Figure 1.

The received signal at the two microphones can be modelled by:

\[ r_1(t) = s(t) + n_1(t), \]
\[ r_2(t) = s(t - D) + n_2(t), \quad 0 \leq t \leq T \quad (1)\]

where \( r_1(t) \) and \( r_2(t) \) are the outputs of two spatially separated microphones, \( s(t) \) is the source signal, \( n_1(t) \) and \( n_2(t) \) represent the additive noises, \( T \) denotes the observation interval, and \( D \) yields the time delay between the two received signals.

The signal and noises are assumed to be uncorrelated having zero-mean and Gaussian distribution.

There are many algorithms to estimate the time delay \( D \) [1]. The cross-correlation (CC) method is one of the basic solutions of the TDE problem [1]. Many other TDE methods develop based on this algorithm. The CC method cross-correlates the microphone outputs and considers the time argument that corresponds to the maximum peak in the output as the estimated time delay. To improve the peak
detection and time delay estimation, various filters, or weighting functions, have been suggested to be used after the cross correlation [2]. The estimated delay is obtained by finding the time-lag that maximizes the cross-correlation between the filtered versions of the two received signals. This technique is called generalized cross-correlation (GCC) [2]. The GCC method, proposed by Knapp and Carter in 1976, is the most popular technique for TDE due to their accuracy and moderate computational complexity. The role of the filter or weighting function in GCC method is to ensure a large sharp peak in the obtained cross-correlation thus ensuring a high time delay resolution. There are many techniques used to select the weighting function; such as the Roth Processor, the Smoothed Coherence Transform (SCOT), the Phase Transform (PHAT), the Eckart Filter, and the Maximum Likelihood (ML) estimator [1, 2]. They are based on maximizing some performance criteria. These correlation-based methods yield ambiguous results when the noises at the two sensors are correlated with the desired signals. To overcome this problem, higher-order statistics methods were employed [1, 3]. There are also some other algorithms used to estimate the time-delay. The matching x and s (MXS) and matching s and x (MSX) methods [4] compare the cross-correlation of the received signals \( r_1(t) \) and \( r_2(t) \) with the autocorrelation of the reference signal \( r_1(t) \). Algorithms based on minimum error: average square difference function (ASDF) and average magnitude difference function (AMDF), seek position of the minimum difference between signals \( r_1(t) \) and \( r_2(t) \)[4]. Adaptive algorithms such as LMS can also be introduced into the TDE [6]. In these algorithms, the delay estimation process is reduced to a filter delay that gives minimal error. Nowadays, many other methods are employed in the TDE, such as, MUSIC [7], ESPRIT [21], and wavelets [8].

However, time-delay estimation is not an easy task because it may face some practical problems, such as, room reverberation, acoustic background noise and the short observation interval. Most of these problems can combine into a case where the signal-to-noise ratio (SNR) is low. How does the low SNR affect the performances of
these methods or what is the SNR level which makes the result of TDE become unreliable? Since the SNR plays an important role in TDE, a SNR threshold is considered as a distinguishable standard between the high and low SNR.

This report is organized as following: section II depicts five TDE methods. Low SNR problem is represented in section III. Section IV demonstrates simulations in MATLAB in simulated and actual noisy environments, and also some discussions are derived in this section. Finally, in section V and VI, acknowledgment and conclusions are made.
II. TDE METHODS

There are several TDE algorithms that have advantages in reducing computation complexity, easing hardware implementation, and increasing precision. Five of these commonly used TDE methods are adopted here.

1. Cross-correlation (CC) method

One common method to estimate the time delay $D$ is to compute the cross correlation function between the received signals at two microphones. Then locate the maximum peak in the output which represents the estimated time delay [1]. The CC can be modelled by:

$$R_{rlr2}(\tau) = E[r_1(t)r_2(t - \tau)]$$  \hfill (2)

$$D_{CC} = \arg\max_{\tau} [R_{rlr2}(\tau)]$$  \hfill (3)

A block diagram of a cross-correlation processor is shown in Figure 2.
2. Phase transform (PHAT) method

A way to sharpen the cross correlation peak is to whiten the input signals by using weighting function, which leads to the so-called generalized cross-correlation technique (GCC). The block diagram of a generalized cross-correlation processor is shown in Figure 3. The PHAT is a GCC procedure which has received considerable attention due to its ability to avoid causing spreading of the peak of the correlation function [16]. This can be expressed mathematically by:

\[ R_{r_1r_2}(\tau) = \int_{-\infty}^{\infty} \psi_p(f) G_{r_1r_2}(f) e^{j2\pi f \tau} df, \quad (4) \]

\[ \psi_p(f) = \frac{1}{|G_{r_1r_2}(f)|} \quad (5) \]

\[ D_p = \arg \max_{\tau} [R_{r_1r_2}(\tau)] \quad (6) \]

where \( G_{r_1r_2}(f) \) is the cross-spectrum of the received signal, \( \psi_p(f) \) is the PHAT weighting function. According to (3), only the phase information is preserved after the cross-spectrum is divided by its magnitude. Ideally (there is no additive noise), this processor approaches a delta function centered at the correct delay.

Figure 3. Generalized Cross-correlation Processor
### 3. Maximum likelihood (ML) method

The ML (is identical to the HT method proposed by Hannan and Thomson [1]) is another important method within the GCC family since it gives the maximum likelihood solution for TDE problem. The ML weighting function $\psi_{ML}(f)$ is chosen to improve the accuracy of the estimated delay by attenuating the signals fed into the correlator in spectral region where the SNR is the lowest. The popularity of ML estimator stems from its relative simplicity of implementation and its optimality under appropriate conditions. Indeed, for uncorrelated Gaussian signal and noise and single path (i.e. no reverberation), the ML estimator of time delay is asymptotically unbiased and efficient in the limit of long observation intervals [12]. The ML method can be represented by:

$$R_{r_1r_2}^{\text{ML}}(\tau) = \int_{-\infty}^{\infty} \psi_{ML}(f) G_{r_1r_2}(f) e^{j2\pi\tau f} df$$ \hspace{1cm} (7)

$$\psi_{ML}(f) = \frac{1}{\left|G_{r_1r_2}(f)\right|^2} \frac{\left|\gamma_{r_1r_2}(f)\right|^2}{1 - \left|\gamma_{r_1r_2}(f)\right|^2}$$ \hspace{1cm} (8)

$$D_{ML} = \arg \max_{\tau} [R_{r_1r_2}^{\text{ML}}(\tau)]$$ \hspace{1cm} (9)

where $\left|\gamma_{r_1r_2}(f)\right|^2 = \frac{\left|G_{r_1r_2}(f)\right|^2}{G_{r_1r_1}(f)G_{r_2r_2}(f)}$ is the magnitude coherency squared, $\psi_{ML}(f)$ is the ML weighting function. The ML contains the fundamental term $\gamma^2/(1-\gamma^2)$. Thus, greater weight is placed on frequency bands that give near-unity coherence. In contrast, frequency bands in which the coherence is near zero are deemphasized [1]. The ML processor weights the cross-spectral phase according to the estimated cross-spectral phase when the variance of the estimated phase error is the least.
4. Average square difference function (ASDF) method

The ASDF method is based on finding the position of the minimum error square between the two received noisy signals and considering this position value as the estimated time delay \([17]\).

\[
R_{\text{ASDF}}(\tau) = \frac{1}{N} \sum_{n=0}^{N-1} [r_1(n) - r_2(n + \tau)]^2 \tag{10}
\]

\[
D_{\text{ASDF}} = \arg \min_{\tau} R_{\text{ASDF}}(\tau) \tag{11}
\]

The favourable performance of the ASDF-based estimator is due to the fact that it gives perfect estimation in the absence of noise while the direct correlation does not \([17]\). From (5), it is apparent that ASDF requires no multiplication, which is the most significant practical advantage over the other methods. Also, this technique does not require the knowledge of the input spectra.

5. Least mean square (LMS) adaptive filter method

The LMS adaptive filter is a finite impulse response (FIR) filter which automatically adapts its coefficients to minimize the mean square difference between its two inputs; a reference signal and a desired signal. It does not require any a priori knowledge of the input spectrum. Assume that \(r_1(n)\) is the reference signal and \(r_2(n)\) is the desired signal. The LMS output \(y_{12}(n)\) can be expressed by:

\[
y_{12}(n) = W_{12}^T(n)X_{12}(n) \tag{12}
\]

where \(T\) denotes transpose and \(X_{12}(n) = [r_1(n), r_1(n-1), \ldots, r_1(n-L+1)]^T\) is the filter state consisting of the most recent samples of the reference signal. The vector \(W_{12}(n)\) is the L-vector of filter weights at instant \(n\). The error output is

\[
e_{12}(n) = r_2 - W_{12}^T(n)X_{12}(n) \tag{13}
\]

The weight vector is updated every sample according to

\[
W_{12}(n + 1) = W_{12}(n) + \mu e_{12}(n)X_{12}^*(n) \tag{14}
\]

where * denotes the complex conjugate and \(\mu\) is a feedback coefficient that controls
the rate of convergence and the algorithm stability. The algorithm adapts the FIR filter to insert a delay equal and opposite to that existing between the two signals. In an ideal situation (no additive noise), the filter weight corresponding to the true delay would be unity and all other weights would be zero [6]. When in the case where the additive noise exists, the filter weight corresponding to the true delay would be the maximum value compared to the other weights.
III LOW SNR PROBLEM and SNR THRESHOLD for TDE

Generally, room reverberation is considered as the main problem for TDE. Moreover, acoustic background noise may further decrease the performance of time-delay estimators. The performance of TDE is always affected by the reverberation in a room [9]. The problem becomes more challenging once room reverberations raise. In a highly reverberant room, all the known TDE methods become unreliable and even fail. Few early studies have investigated the TDE problem in the presence of a few correlated additive echoes [12]. However, the results obtained cannot be used to predict the effects of reverberation on the TDE performance since reverberation consists in the superposition of a very large number of closely spaced echoes, indeed, which are characterized by temporal and spatial correlation. In particular, the quantitative behaviour of the estimator variance for reverberation can be explained naturally in terms of an equivalent signal-to-noise ratio (SNR), which treats the reverberant energy at the microphone output as undesirable noise. Namely, the high level of reverberation causes the low value of SNR [12].

Another problem in TDE is the observation interval. In practice, the spectrum of \( G_{r1r2}(f) \) cannot be obtained from finite observations, so only an estimated \( \hat{G}_{r1r2}(f) \) can be calculated. Consequently, equation (4) can be represented by:

\[
\hat{R}_{r1r2}(\tau) = \int_{-\infty}^{\infty} \psi(f) \hat{G}_{r1r2}(f) e^{j2\pi f \tau} df,
\]

Finite time measurement causes the estimated cross-power spectrum incurs a certain variance, which may affect the accuracy of TDE [10, 11]. It can lead to a large error when the actual observation interval is short. In many cases of practical interest, however, the assumption of long observation interval is inconsistent with other prevailing conditions, such as assumptions of stationary processes and constant delay will only be satisfied over a limited time interval. However, as indicated in [13], there is actually a tradeoff between observation time and SNR in the TDE problem. In the case where SNR is low, the long observation time is required to ensure the accurate \( \hat{G}_{r1r2}(f) \) can be obtained. On contrary, the higher SNR, the shorter observation
interval is needed. Therefore, the two main problems in TDE can be combined into a situation where the SNR is low. There is degradation of performance at low SNR. This is often evidenced as the threshold phenomena in a plot of the variance of the estimates as a function of the SNR in TDE [14, 15]. These thresholds divide the SNR range into several nonoverlapping regions. At high SNR, this is the ambiguity-free mode of operation where differential delay estimation is subjected only to local errors. For very low SNR values, observations are dominated by noise and are essentially unhelpful for TDE.

SIMULATION and DISCUSSION
The adopted five TDE methods are simulated in MATLAB. The reason for choosing MATLAB as the analysis and simulation tool is that it has more flexible choices to support the simulation and is easy to do modification or data recording.

The simulation is carried out in both simulated and actual noisy environments. The MATLAB code is attached in this paper as Appendix-A and Appendix-B.

During the simulation, the source signal ‘mtlb’ (shown in Figure 4) is loaded from a MATLAB database.

![Source Signal](image)

**Figure 4.** The source signal

1. **Using simulated Gaussian noise**

Firstly, the simulation is carried out in simulated noisy environment, where the additive noises $n_1(t)$ and $n_2(t)$ in equation (1) are assumed to be Gaussian. They are uncorrelated and have zero-mean.

In MATLAB, a zero mean Gaussian signal can be generated by using command ‘randn’, whose variance is one. This noise signal is plotted in Figure 5.
Choose the signal-to-noise ratio (SNR) as 16.36dB and the time delay as 243T seconds, where T is the source signal sampling period (in MATLAB, the sampling frequency of signal ‘mtlb’ is 7418Hz, T is 1.3481x10^{-4}s). The cross-correlation results using CC, PHAT and ML are shown in Figure 6. The x-coordinate denotes the time-lag, and the y-coordinate denotes the resulted cross-correlations. From Figure 6, it can be seen that the peak occurs at the actual time delay.

Choose the value 243T as the actual time-delay, using ASDF algorithm, the simulation result is plotted in Figure 7 and the SNR is calculated as 16.38dB, where the insignificant SNR value difference is due to that the Gaussian noise is generated randomly in MATLAB. It is worth noting in Figure 7, the y-coordinate yields the error square of the two difference noisy signal instead of their cross-correlations. It is apparent that the time lag corresponding to the minimum error is the same as the actual time delay.
Figure 6. Cross-correlation using CC, PHAT and ML in a simulated noisy environment

Figure 7. ASDF output in a simulated noisy environment
(c) For reducing the computation of simulation, choosing the value $4T$ as the actual time-delay and using LMS, the resulted filter tap output is plotted in Figure 8. The y-coordinate denotes the filter weight, and x-coordinate denotes its corresponding time lag. As discussed in II.5, the maximum filter weight value takes place at the actual time delay. In this case, the SNR is 16.42dB, again, the random generation of Gaussian noise cause the difference SNR value.

![Figure 8. Filter-tap output using LMS in a simulated noisy environment](image)

2. Using the actual noise signal

For comparison, the simulation is carried out in practical environment. In this situation, the additive noises may no longer be uncorrelated with zero mean. The actual noise is recorded from the real environment; the noise of the working vacuum cleaner, radio broadcasting and mobile phone ring tones can be considered as actual noise in practical condition. The actual noise signal recorded is plotted in Figure 9. Repeat steps 1(a) to 1(c) in previous section using actual noise (SNR is calculated as 22.35dB). The results are plotted as in Figure 10 to Figure12.
(a) Repeat step 1(a) in previous section but using actual noise. The resulted cross-correlations using CC, PHAT and ML methods in the actual noisy environment are plotted in Figure 10. It is clear that the peak position corresponds to the actual time delay.

Figure 9. An actual noise signal from a vacuum cleaner

Figure 10. Cross-correlations using CC, PHAT and ML in an actual noisy environment
(b) Repeat step 1(b) in previous section but using actual noise. As expected, the time delay is estimated as $243T$ using ASDF algorithm in the actual noisy environment. Figure 11 shows the simulation result.

![Figure 11. ASDF output in an actual noisy environment](image)

(c) Repeat step 1(c) in previous section but using actual noise. The estimated time delay $4T$ can be obtained using LMS methods in the actual noisy environment. Figure 12 shows the resulted filter tap output.

![Figure 12. Filter-tap output using LMS in an actual noisy environment](image)
3. The relation between the SNR level and the time delay estimation accuracy using Gaussian noise

In this section, the time delay is estimated under various SNR levels. The noise type is Gaussian with zero mean and variance is equal to one.

To study the performance of the time delay estimation, the following experiments have been set.

(a) The actual time delay value is set to $243T$, we calculate the time delay using CC, PHAT and ML in different SNR situations. The various SNR level is obtained by altering the noise power. The results are plotted in Figure 13. The x-coordinate presents the various SNR values, while the y-coordinate presents the estimated time delay. The plots show that the estimated time delay becomes incorrect when the SNR exceeds a certain threshold. In this case, SNR thresholds are about -15dB, -11dB and -11.5dB for CC, PHAT and ML, respectively.

![Figure 13. The relations between SNR and estimated time delay using CC, PHAT and ML in a simulated environment](image)
(b) The time delay is set to 243T. The time delay is estimated using ASDF method with various SNR level and the result is depicted in Figure 14. The SNR threshold beyond which the detection fails is around -13dB.

![Figure 14. The relation between SNR and estimated time delay using ASDF in a simulated environment](image)

(c) During the simulation of LMS, varying SNR level, it can be obtained that when SNR is below 5.09dB, the estimated time delay becomes inaccurate.

4. **The relation between the SNR and accuracy of time delay estimation under actual noise environment**

   In this section, the time delay is estimated with different SNR level, and the actual noise is adopted as the additive noise.

   (a) Repeat step 3(a) in the previous section but using actual noise. The results are plotted as in Figure 15. It shows that SNR thresholds change to about -3dB, -18dB and -10dB for CC, PHAT and ML, respectively.
(d) Repeat step 3(b) in the previous section but using actual noise. The ASDF performance under noise environment is shown in Figure 16, the SNR threshold increase to about -2dB.
(c) Repeat step 3(c) in the previous section but using actual noise. Again, according to the simulation of LMS, it can be obtained that when SNR is below -8.72dB, the estimated time delay becomes inaccurate.

The computer simulation results presented here are useful to demonstrate the relative accuracy of different time delay estimates.

According to Figure 6 and Figure 10, it is apparent that the resulted cross-correlations of GCC methods (in this case, they are PHAT and ML) show sharper peaks at the correct time delay compared to the CC result in both simulated and actual noisy environments. They actually accentuate the correct differential time delays by suppressing the adjacent peaks in the observed cross-correlation functions. However, the PHAT method has a better performance in sharpening the cross-correlation peak compared to the other GCC methods in both simulated and actual noisy environments. Indeed, without noise, the result cross-correlation of PHAT will be a delta function in the correct time delay.

From Figure 7 and Figure 11, both in simulated and actual noisy environments, the estimated time delay using ASDF method also can be detected easily due to its sharp valley.

All the five TDE methods show excellent results in high SNR environments. However, their performances become very poor when SNR is decreased. In this situation, the noisy signal becomes noise domain and a big distortion produced by the additive noise presents in the crosspower spectrum, thus the prefilter attempts to whiten the noise rather than the source signal. As expected, from Figures 13-16, each method has its own SNR threshold. Above this threshold, the estimated time delay becomes accurate and unbiased. On the other hand, the estimation method fails to function properly below the SNR threshold. From Figures 13-16, it seems that CC and ASDF have the smaller SNR threshold in simulated noisy conditions, while PHAT has the smallest SNR threshold in actual noisy conditions. This indicates that CC and ASDF outperform the other TDE methods at low SNR conditions in simulated noisy environments, however PHAT has the best performance in actual environment.
The reason that SNR thresholds increase when the environment changes from the simulated noisy condition to the actual noisy condition is described as following:

Mathematically, the cross-correlation of the two noisy signals \( r_1(t) \) and \( r_2(t) \) in equation (1) can be written as:

\[
R_{r_1 r_2}(\tau) = R_{s,s}(\tau) + R_{s,n_1}(\tau) + R_{s,n_2}(\tau) + R_{n_1 n_2}(\tau) \tag{16}
\]

The signal and the noise are assumed to be uncorrelated. Thus \( R_{s,n_1}(\tau) \), \( R_{s,n_2}(\tau) \) are zero. Also, the additive noises are assumed uncorrelated, thus, the value of the last term in (10) is zero. However, the last assumption is not true practically, it cannot be neglected due to the existing correlation between the two additive noise signals. This kind of correlation obviously affects the result of time delay estimation. That is why for most TDE methods, SNR thresholds should be increased when the environment changes from simulated noisy condition to actual noisy condition. As discussed in section II.2, since only the phase information is preserved after perfiltering the received signal using PHAT filter, the PHAT method is not seriously affected by the correlation of the additive noises, thus when the environment changes, the PHAT SNR threshold does not vary as same as the other TDE methods do, which can be seen from the Figure 13 and Figure 15.

The algorithms to estimate the time delay introduced so far can only be used to compute the time delays having an integer number of samples. Thus when we are interested in total delay \( D \) which consists also of fractional part \( \hat{d} \), Lagrange [5] or parabolic [20] interpolation methods can be used. (e.g. parabolic interpolation which results in following equation [20].)

\[
D = \hat{D} + \hat{d} = \hat{D} - 0.5 \frac{R[\hat{D}+1] - R[\hat{D}-1]}{R[\hat{D}+1] - 2R[\hat{D}] + R[\hat{D}-1]} \tag{17}
\]

where \( \hat{D} \) is the estimated integer delay using any of the described algorithms above.
V. CONCLUSIONS

The five TDE methods described in this report work well in case of high SNR. In the simulated noisy environment, ASDF should be adopted due to its simple computation, easy detection and small SNR threshold. However, in the actual noisy environment, PHAT seems to be the best choice because of its perfect performance in sharpening the correlation at the correct time delay and its small SNR threshold. However, when the SNR is below a specified threshold, the performance of all the five methods rapidly deteriorates due to large anomalous or ambiguous estimates. While the anomaly and ambiguity effects are fundamental and unavoidable features of the delay estimation problem, independent of the signal processing technique, which make the TDE problem more challenging when the SNR drops under a minimal level. Future work should be focused on the robust time delay estimation with low SNR.
VI. REFERENCES


Appendix A

It is worth noting that when the reader run the following programmes, they may not obtain the exactly same result as the above figures due to the Gaussian noise is generated randomly in MATLAB.

1. time delay estimation using CC, PHAT and ML in Gaussian noise
% this programme is to estimate the time delay using CC,PHAT and ML
% using Gaussian noise with zero mean and one variance
% generate a speech signal, and zero pattern its length so that it is
% suitable for FFT.
load mtlb;
l1=length(mtlb);
b=1;
l2=b*l1;
signal=zeros(l2,1);
for m=1:b;
    for n=1:l1;
        signal(n+(m-1)*l1)=mtlb(n);
    end
end
signallength=length(signal);
p=1;
for n=1:signallength;
    if 2.^n<(signallength*2);
        p=p+1;
    end
end
assumesignallength=2.^p;
signal1=zeros(assumesignallength,1);
for n=1:signallength;
    signal1(n)=signal(n);
end

% assume a time delay value;
delay=243;

% generate a time delayed speech signal;
signal2=lagmatrix(signal1,delay);
signal2(1:delay)=0;

% generate noise signals;
noise1=randn(signallength,1);
noise2=randn(signallength,1);

for n=(signallength+1):assumesignallength;
    noise1(n)=0;
    noise2(n)=0;
end

signal1=signal1*10;
signal2=signal2*10;

% generate the noisysignal by adding the speech signal and noise signal;
noisysignal1=signal1+noise1;
noisysignal2=signal2+noise2;

% calculate the signal-to-noise ratio;
signalpower=0;
noisepower=0;

for n=1:assumesignallength;
    signalpower=signalpower+(abs(signal1(n)))^2;
    noisepower=noisepower+(abs(noise1(n)))^2+(abs(noise2(n)))^2;
end

signalpower=signalpower/signallength;
noisepower=noisepower/(signallength*2);

snr=10*log10(signalpower/noisepower);
% calculate the crosscorrelation of two noisy signal;
cccorrelation=xcorr(noisysignal1,noisysignal2);
% CC method: find the peak position of the crosscorrelation;
[ccmaximum,cctime]=max(cccorrelation);
cestimation=abs(assumesignallength-cctime);
% GCC method: using the PHAT filter;
gcc=zeros((assumesignallength*2-1),1);
phatfilter=zeros((assumesignallength*2-1),1);
crossspectrum=fft(cccorrelation);
for n=1:(assumesignallength*2-1);
    phatfilter(n)=abs(crossspectrum(n));
    gcc(n)=crossspectrum(n)/phatfilter(n);
end
gcccorrelation=ifft(gcc);
% GCC method: find the peak position of the filtered crosscorrelation;
for n=1:(assumesignallength*2-1);
    gcccorrelation(n)=abs(gcccorrelation(n));
end
[gccmaximum,gctime]=max(gcccorrelation);
gccestimation=abs(assumesignallength-gctime);
% Maximum Likelihood method: using the ML filter;
coherence=cohere(noisysignal1,noisysignal2);
mlcoherence=zeros((assumesignallength*2-1),1);
coherencelength=length(coherence);
coherencenumber=(assumesignallength*2-1)/coherencelength;
p=fix(coherencenumber);
if p<coherencenumber;
    for n=1:coherencelength;
        for m=1:p;
            mlcoherence(m+(n-1)*p)=coherence(n);
        end
    end
end
for n=((coherencelength*p+1):(assumesignallength*2-1));
    mlcoherence(n)=coherence(coherencelength);
end
if p==coherencenumber;
    for n=1:coherencelength;
        for m=1:coherencenumber;
            mlcoherence(m+(n-1)*coherencenumber)=coherence(n);
        end
    end
end
for n=1:(assumesignallength*2-1);
    squaremlcoherence(n)=(abs(mlcoherence(n))).^2;
    ml(n)=squaremlcoherence(n)*crossspectrum(n)/(phatfilter(n)*(1-squaremlcoherence(n)));
end
mlcorrelation=ifft(ml);
% ML method: find the peak position of ML correlation;
for n=1:(assumesignallength*2-1);
    mlcorrelation(n)=abs(mlcorrelation(n));
end
[mlmaximum,mltime]=max(mlcorrelation);
mlestimation=abs(assumesignallength-mltime);
% show the SNR, and the three estimated time delay value;
SNR,ccestimation,gccestimation,mlestimation
% plot the three crosscorrelation;
lag=zeros((assumesignallength*2-1),1);
for n=1:(assumesignallength*2-1);
lag(n)=assumesignallength-n;

end
maximum=delay+50;
minimum=delay-50;
subplot(3,1,1);
plot(lag,cccorrelation,'b')
axis([minimum maximum -inf inf]);
legend('CC');
subplot(3,1,2);
plot(lag,gcccorrelation,'r')
axis([minimum maximum -inf inf]);
ylabel('cross-correlation');
legend('PHAT');
subplot(3,1,3);
plot(lag,mlcorrelation,'g')
axis([minimum maximum -inf inf]);
xlabel('time lag');
legend('ML');

2. time delay using ASDF in Gaussian noise
% this programme is to estimate the time-delay using ASDF
% using Gaussian noise with zero mean and one variance
% generate a speech signal;
load mtlb;

signallength=length(mtlb);
assumesignallength=2*signallength;
signal1=zeros(assumesignallength,1);
for n=1:signallength;
    signal1(n)=mtlb(n);
end
% assume a time-delay;
delay=243;
% generate a time-delay signal;
signal2=lagmatrix(signal1,delay);
signal2(1:delay)=0;
% generate a noise signal;
noise1=randn(signallength,1);
noise2=randn(signallength,1);
for n=(signallength+1):assumesignallength;
    noise1(n)=0;
    noise2(n)=0;
end
% generate two noisysignal by adding the signal and noise;
signal1=signal1*10;
signal2=signal2*10;
noisysignal1=signal1+noise1;
noisysignal2=signal2+noise2;
% calculate the signal-to-noise ration, SNR
signalpower=0;
noisepower=0;
for n=1:assumesignallength;
    signalpower=signalpower+(abs(signal1(n)))^2;
    noisepower=noisepower+(abs(noise1(n)))^2+(abs(noise2(n)))^2;
end
signalpower=signalpower/signallength;
noisepower=noisepower/(signallength*2);
SNR=10*log10(signalpower/noisepower);
% Average Square Difference function method;
% change the lag;
for lag=1:400;
    delaynoisysignal1=lagmatrix(noisysignal1,lag);
    delaynoisysignal1(1:lag)=0;
    error(lag)=0;
% calculate the square error;
    for n=1:assumesignallength;
        error(lag)=error(lag)+(delaynoisysignal1(n)-noisysignal2(n))^2;
    end
end
% find the lag of the minimum error square;
[minimum estiamtion]=min(error);

3. time delay estimation using CC, PHAT and ML in actual noise
% this programme is to estimate the time delay using CC,PHAT and ML
% using actual noise
% generate a speech signal;
load mtlb;
l1=length(mtlb);
b=1;
l2=b*l1;
signal=zeros(l2,1);
for m=1:b;
    for n=1:l1;
        signal(n+(m-1)*l1)=mtlb(n);
    end
end
end
signallength=length(signal);
p=1;
for n=1:signallength;
    if 2.^n<(signallength*2);
        p = p + 1 ;
    end
end
assumesignallength=2.^p;
signal1=zeros(assumesignallength,1);
for n=1:signallength;
    signal1(n)=signal(n);
end
% assume a time delay value;
delay=234;
% generate a time delay speech signal;
signal2=lagmatrix(signal1,delay);
signal2(1:delay)=0;
% generate a noise signal;
a=wavread('actualnoise.wav');
noise1=a(1:assumesignallength);
noise2=a(10001:(assumesignallength+10000));
for n=(signallength+1):assumesignallength;
    noise1(n)=0;
    noise2(n)=0;
end
signal1=signal1;
signal2=signal2;
% generate the noisysignal by adding the speech signal and noise signal;
noisysignal1=signal1+noise1;
noisysignal2 = signal2 + noise2;

% calculate the signal-to-noise ratio;
signalpower = 0;
noisepower = 0;
for n = 1:assumesignallength;
    signalpower = signalpower + (abs(signal1(n)))^2;
    noisepower = noisepower + (abs(noise1(n)))^2 + (abs(noise2(n)))^2;
end
signalpower = signalpower / signallength;
oisepower = noisepower / (signallength * 2);

snr = 10 * log10(signalpower / noisepower);

% calculate the crosscorrelation of two noisy signal;
cccorrelation = xcorr(noisysignal1, noisysignal2);

% CC method: find the peak position of the crosscorrelation;
[ccmaximum cctime] = max(cccorrelation);
ccestimation = abs(assumesignallength - cctime);

% GCC method: using the PHAT filter;
gcc = zeros((assumesignallength * 2 - 1), 1);
phatfilter = zeros((assumesignallength * 2 - 1), 1);
crossspectrum = fft(cccorrelation);
for n = 1:(assumesignallength * 2 - 1);
    phatfilter(n) = abs(crossspectrum(n));
    gcc(n) = crossspectrum(n) / phatfilter(n);
end
gcccorrelation = ifft(gcc);

% GCC method: find the peak position of the filtered crosscorrelation;
for n = 1:(assumesignallength * 2 - 1);
    gcccorrelation(n) = abs(gcccorrelation(n));
end
[gccmaximum, gcctime] = max(gcccorrelation);
gccEstimation = abs(assumesignalLength - gcctime);

% Maximum Likelihood method: using the ML filter;
coherence = cohere(noisySignal1, noisySignal2);
mlCoherence = zeros((assumesignalLength * 2 - 1), 1);
coherenceLength = length(coherence);
coherenceNumber = (assumesignalLength * 2 - 1) / coherenceLength;
p = fix(coherenceNumber);
if p < coherenceNumber;
    for n = 1:coherenceLength;
        for m = 1:p;
            mlCoherence(m + (n-1)*p) = coherence(n);
        end
    end
end
for n = ((coherenceLength * p + 1):(assumesignalLength * 2 - 1));
    mlCoherence(n) = coherence(coherenceLength);
end
if p == coherenceNumber;
    for n = 1:coherenceLength;
        for m = 1:coherenceNumber;
            mlCoherence(m + (n-1)*coherenceNumber) = coherence(n);
        end
    end
end
for n = 1:(assumesignalLength * 2 - 1);
    squareMLCoherence(n) = (abs(mlCoherence(n))).^2;
    ml(n) = squareMLCoherence(n) * crossspectrum(n) / (phatFilter(n) * (1-squareMLCoherence(n)));
end
mlCorrelation = ifft(ml);
% ML method: find the peak position of ML correlation;
for n=1:(assumesignal_length*2-1);
    mlcorrelation(n)=abs(mlcorrelation(n));
end
[mlmaximum,mltime]=max(mlcorrelation);
mlestimation=abs(assumesignal_length-mltime);
% show the SNR, and the three estimated time delay value;
SNR, ccestimation, gccestimation, mlestimation
% plot the three crosscorrelation;
lag=zeros((assumesignal_length*2-1),1);
for n=1:(assumesignal_length*2-1);
    lag(n)=assumesignal_length-n;
end
maximum=delay+50;
minimum=delay-50;
subplot(3,1,1);
plot(lag,cccorrelation,'b')
axis([minimum maximum -inf inf]);
legend('CC');
subplot(3,1,2);
plot(lag,geccorrelation,'r')
axis([minimum maximum -inf inf]);
ylabel('cross-correlation');
legend('PHAT');
subplot(3,1,3);
plot(lag,mlcorrelation,'g')
axis([minimum maximum -inf inf]);
legend('ML');
xlabel('time lag');
4. **time delay estimation using ASDF in actual noise**

% this programme is to estimate the time delay using ASDF
% using actual noise
% generate a speech signal;
load mtlb;
signallength=length(mtlb);
p=1;
for n=1:signallength;
    if 2.^n<(signallength*2);
        p=p+1;
    end
end
assumesignallength=2.^p;
signal1=zeros(assumesignallength,1);
signal1=lagmatrix(signal1,delay);
signal2(1:delay)=0;
% generate a noise signal;
a=wavread('actualnoise.wav');
noise1=a(1:signallength);
noise2=a(10001:(signallength+10000));
for n=(signallength+1):assumesignallength;
    noise1(n)=0;
    noise2(n)=0;
end

% generate two noisysignal by adding the signal and noise;
signal1=signal1*1;
signal2=signal2*1;
noisysignal1=signal1+noise1;
noisysignal2=signal2+noise2;

% calculate the signal-to-noise ratio, SNR
signalpower=0;
noisepower=0;
for n=1:assumesignallength;
  signalpower=signalpower+(abs(signal1(n)))^2;
  noisepower=noisepower+(abs(noise1(n)))^2+(abs(noise2(n)))^2;
end
signalpower=signalpower/signallength;
noisepower=noisepower/(signallength*2);

snr=10*log10(signalpower/noisepower);

% Average Square Difference function method;
% change the lag;
for lag=1:400;
  delaynoisysignal1=lagmatrix(noisysignal1,lag);
  delaynoisysignal1(1:lag)=0;
  error(lag)=0;
  for n=1:assumesignallength;
    error(lag)=error(lag)+(delaynoisysignal1(n)-noisysignal2(n))^2;
  end
end

% find the lag of the minimum error square;
[minimum estimation]=min(error);

snr,estimation
plot(error)
xlabel('time lag')
ylabel('error square')

5. the estimated time delay versus SNR using CC, PHAT and ML in Gaussian noise operation model

% this programme is to compare SNR and the estimated time delay using CC
% PHAT and ML in Gaussian random noise with zero mean and one variance
% generate a speech signal;
load mtlb;
l1=length(mtlb);
b=1;
l2=b*l1;
originalsignal=zeros(l2,1);
for m=1:b;
    for n=1:l1;
        originalsignal(n+(m-1)*l1)=mtlb(n);
    end
end
signallength=length(originalsignal);
p=1;
for n=1:signallength;
    if 2.^n<(signallength*2);
        p=p+1;
    end
end
assumesignallength=2.^p;
originalsignal1=zeros(assumesignallength,1);
for n=1:signallength;
originalsignal1(n)=originalsignal(n);

end

% assume a time delay value;
delay=243;
% generate a time delay signal;
originalsignal2=lagmatrix(originalsignal1,delay);
originalsignal2(1:delay)=0;
% generate a noise signal;
noise1=randn(signallength,1);
noise2=randn(signallength,1);
for n=(signallength+1):assumesignallength;
    noise1(n)=0;
    noise2(n)=0;
end
% change the signal power, get the various signal-to-noise ratio;
scale=0.2:0.02:1;
scalenum= length(scale);
snr=zeros(1,scalenum);
for t=1:scalenum;
    signal1=originalsignal1*scale(t);
    signal2=originalsignal2*scale(t);
    noisysignal1=signal1+noise1;
    noisysignal2=signal2+noise2;
    signalpower=0;
    noisepower=0;
    for n=1:assumesignallength;
        signalpower=signalpower+(abs(signal1(n)))^2;
        noisepower=noisepower+(abs(noise1(n)))^2+(abs(noise2(n)))^2;
    end
    signalpower=signalpower/signallength;
end
noisepower=noisepower/(signallength*2);
snr(t)=10*log10(signalpower/noisepower);

% CC method;
cccorrelation=xcorr(noisysignal1,noisysignal2);
[ccmaximum cctime]=max(cccorrelation);
ccestimation(t)=abs(assumesignallength-cctime);

% GCC method: using the PHAT filter;
gcc=zeros((assumesignallength*2-1),1);
phatfilter=zeros((assumesignallength*2-1),1);
crossspectrum=fft(cccorrelation);
for n=1:(assumesignallength*2-1);
    phatfilter(n)=abs(crossspectrum(n));
    gcc(n)=crossspectrum(n)/phatfilter(n);
end
gcccorrelation=ifft(gcc);
for n=1:(assumesignallength*2-1);
    gcccorrelation(n)=abs(gcccorrelation(n));
end
[gccmaximum,gctime]=max(gcccorrelation);
gccestimation(t)=abs(assumesignallength-gctime);

% ML method;
coherence=cohere(noisysignal1,noisysignal2);
mlcoherence=zeros((assumesignallength*2-1),1);
coherence_length=length(coherence);
coherence_number=(assumesignallength*2-1)/coherence_length;
p=fix(coherence_number);
if p<coherence_number;
    for n=1:coherence_length;
        for m=1:p;
            mlcoherence(m+(n-1)*p)=coherence(n);
        end
    end
end
end

end

for n=((coherencelength*p+1):(assumesignallength*2-1));
    mlcoherence(n)=coherence(coherencelength);
end

if p==coherencenumber;
    for n=1:coherencelength;
        for m=1:coherencenumber;
            mlcoherence(m+(n-1)*coherencenumber)=coherence(n);
        end
    end
end

for n=1:(assumesignallength*2-1);
    squaremlcoherence(n)=(abs(mlcoherence(n))).^2;
end

ml(n)=squaremlcoherence(n)*crossspectrum(n)/(phatfilter(n)*(1-squaremlcoherence(n)));

mlcorrelation=ifft(ml);

mlcorrelation=ifft(ml);

for n=1:(assumesignallength*2-1);
    mlcorrelation(n)=abs(mlcorrelation(n));
end

[mlmaximum,mltime]=max(mlcorrelation);

mlestimation(t)=abs(assumesignallength-mltime);

% plot the time-delay value versus SNR;
subplot(3,1,1);
stem(snr,ccestimation,'b')
legend('CC');
subplot(3,1,2);
stem(snr,gceestimation,'r')
ylabel('estimated time-delay');
legend('PHAT');
subplot(3,1,3);
stem(snr,mlestimation,'g');
legend('ML');
xlabel('SNR');

6. **the estimated time delay versus SNR using ASDF in Gaussian noise operation model**

% this programme is to compare SNR and estimated time delay using ASDF in
% Gaussian random noise with zero mean and one variance
% generate a speech signal;
load mtlb;
l1=length(mtlb);
b=1;
l2=b*l1;
originalsignal=zeros(l2,1);
for m=1:b;
    for n=1:l1;
        originalsignal(n+(m-1)*l1)=mtlb(n);
    end
end
signallength=length(originalsignal);
p=1;
for n=1:signallength;
    if 2.^n<(signallength*2);
        p=p+1;
    end
assumesignallength=2.^p;
originalsignal1=zeros(assumesignallength,1);
for n=1:signallength;
    originalsignal1(n)=originalsignal(n);
end
% assume a time-delay;
delay=243;
% generate a time-delay speech signal;
originalsignal2=lagmatrix(originalsignal1,delay);
originalsignal2(1:delay)=0;
% generate a noise signal;
oise1=randn(signallength,1);
oise2=randn(signallength,1);
for n=(signallength+1):assumesignalength;
    noise1(n)=0;
    noise2(n)=0;
end
% change the SNR by varying the signal power;
scale=0.1:0.02:1;
t=length(scale);
for m=1:t;
    signal1=originalsignal1*scale(m);
signal2=originalsignal2*scale(m);
noisysignal1=signal1+noise1;
oisysignal2=signal2+noise2;
signalpower=0;
noisepower=0;
for n=1:assumesignallength;
    signalpower=signalpower+(abs(signal1(n)))^2;
end
noisepower=noisepower+(abs(noise1(n)))^2+(abs(noise2(n)))^2;
end
signalpower=signalpower/signallength;
noisepower=noisepower/(signallength*2);

snr(m)=10*log10(signalpower/noisepower);

% Average Square Difference Function method (ASDF);
for lag=1:400;
  delaynoisysignal1=lagmatrix(noisysignal1,lag);
  delaynoisysignal1(1:lag)=0;
  error(lag)=0;
  for n=1:assumesignallength;
    error(lag)=error(lag)+(delaynoisysignal1(n)-noisysignal2(n))^2;
  end
end

[minimum estimation]=min(error);

asdfestimation(m)=estimation;
end

% plot the estimated time delay versus SNR
stem(snr,asdfestimation)
xlabel('SNR');
ylabel('estimated time-delay');

7. the estimated time delay versus SNR using CC, PHAT and ML in the actual noise operation model

% this programme is to compare SNR and the estimated time delay using CC
% PHAT and ML in the actual noise condition
% generate a speech signal;
load mtlb;
l1=length(mtlb);
b=1;
l2=b*l1;
originalsignal=zeros(l2,1);
for m=1:b;
    for n=1:l1;
        originalsignal(n+(m-1)*l1)=mtlb(n);
    end
end
signallength=length(originalsignal);
p=1;
for n=1:signallength;
    if 2.^n<(signallength*2);
        p = p + 1;
    end
end
assumesignallength=2.^p;
originalsignal1=zeros(assumesignallength,1);
for n=1:signallength;
    originalsignal1(n)=originalsignal(n);
end
% assume a time delay value;
delay=243;
% generate a time delay signal;
originalsignal2=lagmatrix(originalsignal1,delay);
originalsignal2(1:delay)=0;
% generate a noise signal;
a wavread('actualnoise.wav');
noise1=a(1:signallength);
noise2=a(10001:(signallength+10000));
for n=(signallength+1):assumesignallength;
noise1(n)=0;
noise2(n)=0;
end

% change the signal power, get the various signal-to-noise ratio;
scale=0.004:0.002:0.1;
scalenumber=length(scale);

snr=zeros(1,scalenumber);
for t=1:scalenumber;
    signal1=originalsignal1*scale(t);
    signal2=originalsignal2*scale(t);
    noisysignal1=signal1+noise1;
    noisysignal2=signal2+noise2;
    signalpower=0;
    noisepower=0;
    for n=1:assumesignallength;
        signalpower=signalpower+(abs(signal1(n)))^2;
        noisepower=noisepower+(abs(noise1(n)))^2+(abs(noise1(n)))^2;
    end
    signalpower=signalpower/signallength;
    noisepower=noisepower/(signallength*2);
    snr(t)=10*log10(signalpower/noisepower);
end

% CC method;
cccorrelation=xcorr(noisysignal1,noisysignal2);
[ccmaximum cctime]=max(cccorrelation);
ccestimation(t)=abs(assumesignallength-cctime);

% GCC method: using the PHAT filter;
gcc=zeros((assumesignallength*2-1),1);
phatfilter=zeros((assumesignallength*2-1),1);
crossspectrum=fft(cccorrelation);
for n=1:(assumesignallength*2-1);
\[
\text{phatfilter}(n) = \text{abs(crossspectrum}(n));
\]
\[
\text{gcc}(n) = \text{crossspectrum}(n)/\text{phatfilter}(n);
\]
end
\[
\text{gcccorrelation} = \text{ifft(gcc)};
\]
for \( n = 1 : (\text{assumesignallength} \times 2 - 1) \)
\[
\text{gcccorrelation}(n) = \text{abs(gcccorrelation}(n));
\]
end
\[
[\text{gccmaximum,gcctime}] = \text{max(gcccorrelation)};
\]
\[
\text{gccestimation}(t) = \text{abs(assumesignallength-gcctime)};
\]
\% ML method;
\[
\text{coherence} = \text{cohere(noisysignal1, noisysignal2)};
\]
\[
\text{mlcoherence} = \text{zeros}((\text{assumesignallength} \times 2 - 1),1);
\]
\[
\text{coherence} = \text{length(coherence)};
\]
\[
\text{coherence} = (\text{assumesignallength} \times 2 - 1)/\text{coherence};
\]
\[
\text{p} = \text{fix(coherence));
\]
if \( p < \text{coherence} \)
\[ 
\text{for} \ n = 1: \text{coherence} \]
\[ 
\text{for} \ m = 1:p;
\]
\[ 
\text{mlcoherence}(m+(n-1)*p) = \text{coherence}(n);
\]
end
end
end
end
\[ 
\text{for} \ n = (\text{coherence} \times p + 1): (\text{assumesignallength} \times 2 - 1) \]
\[ 
\text{mlcoherence}(n) = \text{coherence} (\text{coherence}) ;
\]
end
\[ 
\text{if} \ p == \text{coherence} ;
\]
\[ 
\text{for} \ n = 1: \text{coherence} ;
\]
\[ 
\text{for} \ m = 1: \text{coherence} ;
\]
\[ 
\text{mlcoherence}(m+(n-1)*\text{coherence}) = \text{coherence} (n); 
\]
end
end

end

for n=1:(assumesignallength*2-1);
    squaremlcoherence(n)=(abs(mlcoherence(n))).^2;
end

ml(n)=squaremlcoherence(n)*crossspectrum(n)/(phatfilter(n)*(1-squaremlcoherence(n)));end

mlcorrelation=ifft(ml);

for n=1:(assumesignallength*2-1);
    mlcorrelation(n)=abs(mlcorrelation(n));
end

[mlmaximum,mltime]=max(mlcorrelation);
mlestimation(t)=abs(assumesignallength-mltime);
end

% plot the time-delay value versus SNR;
subplot(3,1,1);
stem(snr,ccestimation,'b')
legend('CC');
subplot(3,1,2);
stem(snr,gccestimation,'r')
ylabel('estimated time-delay')
legend('PHAT');
subplot(3,1,3);
stem(snr,mlestimation,'g')
legend('ML');
xlabel('SNR');
8. the estimated time delay versus SNR using ASDF in the actual noise operation model

% this programme is to compare SNR and the estimated time delay using ASDF
% in the actual noise condition
% generate a speech signal;
load mtlb;
l1=length(mtlb);
b=1;
l2=b*l1;
originalsignal=zeros(l2,1);
for m=1:b;
    for n=1:l1;
        originalsignal(n+(m-1)*l1)=mtlb(n);
    end
end
signallength=length(originalsignal);
p=1;
for n=1:signallength;
    if 2.^n<(signallength*2);
        p=p+1;
    end
end
assumesignallength=2.^p;
originalsignal1=zeros(assumesignallength,1);
for n=1:signallength;
    originalsignal1(n)=originalsignal(n);
end
% assume a time-delay;
delay=243;
% generate a time-delay speech signal;
originalsignal2 = lagmatrix(originalsignal1, delay);
originalsignal2(1:delay) = 0;

% generate a noise signal;
a = wavread('actualnoise.wav');
noise1 = a(1:signallength);
noise2 = a(10001:(signallength + 10000));
for n = (signallength + 1):assumesignallength;
    noise1(n) = 0;
    noise2(n) = 0;
end

% change the SNR by varying the signal power;
scale = 0.03:0.005:0.13;
t = length(scale);
for m = 1:t;
    signal1 = originalsignal1 * scale(m);
    signal2 = originalsignal2 * scale(m);
    noisysignal1 = signal1 + noise1;
    noisysignal2 = signal2 + noise2;
    signalpower = 0;
    noisepower = 0;
    for n = 1:assumesignallength;
        signalpower = signalpower + (abs(signal1(n)))^2;
        noisepower = noisepower + (abs(noise1(n)))^2 + (abs(noise2(n)))^2;
    end
    signalpower = signalpower / signallength;
    noisepower = noisepower / (signallength * 2);
    snr(m) = 10 * log10(signalpower / noisepower);
% Average Square Difference Function method (ASDF);
for lag = 1:400;
    delaynoisysignal1 = lagmatrix(noisysignal1, lag);
end
delaynoisysignal1(1:lag)=0;
error(lag)=0;
for n=1:assumesignallength;
    error(lag)=error(lag)+(delaynoisysignal1(n)-noisysignal2(n))^2;
end
end
[minimum estimation]=min(error);
asdfestimation(m)=estimation;
end

% plot the estimated time delay versus SNR
stem(snr,asdfestimation)
xlabel('SNR');
ylabel('estimated time-delay');
Appendix B

1. Time delay estimation using LMS in Gaussian random noise

% this programme should be run before the simulation of LMS to obtain the % speech and Guassian noise signal in simulink of LMS

load mtlb;
speechsignal=10*mtlb;
noise1=randn(4001,1);
noise2=randn(4001,1);
source=zeros(4001,2);
n1=zeros(4001,2);
n2=zeros(4001,2);
for n=1:4001;
    source(n,1)=n;
    source(n,2)=speechsignal(n);
    n1(n,1)=n;
    n1(n,2)=noise1(n);
    n2(n,1)=n;
    n2(n,2)=noise2(n);
end
signalpower=0;
noisepower=0;
for n=1:4001;
    signalpower=signalpower+(abs(speechsignal(n)))^2;
    noisepower=noisepower+(abs(noise1(n)))^2+(abs(noise2(n)))^2;
end
signalpower=signalpower/4001;
noisepower=noisepower/8002;
snr=10*log10(signalpower/noisepower);
snr
2. Time delay estimation using LMS in actual noise

% this programme should be run before the simulation of LMS to obtain the
% speech and actual noise signal in simulink of LMS

load mtlb;
speechsignal=mtlb;
a=wavread('actualnoise.wav');
noise1=a(1:4001);
noise2=a(10001:14001);
source=zeros(4001,2);
n1=zeros(4001,2);
n2=zeros(4001,2);
for n=1:4001;
    source(n,1)=n;
    source(n,2)=speechsignal(n);
    n1(n,1)=n;
    n1(n,2)=noise1(n);
    n2(n,1)=n;
    n2(n,2)=noise2(n);
end
signalpower=0;
noisepower=0;
for n=1:4001;
    signalpower=signalpower+(abs(speechsignal(n)))^2;
    noisepower=noisepower+(abs(noise1(n)))^2+(abs(noise2(n)))^2;
end
signalpower=signalpower/4001;
noisepower=noisepower/8002;
snr=10*log10(signalpower/noisepower);
snr