DSPLAB Project 2: Implementation and Application of Wiener Filter and LMS

I. Goal

To learn the theory of the Wiener filter, Least Mean Square (LMS) algorithm, and the use of the Room Impulse Response Generator (RIRG) tool. Conduct simulations in Matlab and discuss the results.

II. Experimental Tools

wienerscalart96.m	The m-file for the Wiener Filter can be used to eliminate noise by inputting an audio file and other parameters.
audio_SNR.m	Input clean speech and noise to generate mixed speech.
Noise.wav*3	Three different types of noise: horn, mouse, and white noise.
clean speech.wav*10	Ten segments of clean speech.
echo.m	Input an audio file and other parameters to simulate the echo generated in a room.
LMS.m	Input an audio file and other parameters to eliminate echo.
1.wav	A segment of clean speech.

III. Experimental Procedure

Experiment One

Step 1 : **Practice completing the program for mixing clean speech with noise.**

Please completing the 'audio_SNR.m' program according to the comments next to the program, so that it can mix clean speech with noise.

Hint: Signal-to-Noise Ratio (SNR): the ratio of signal power to noise power.

SNR (Signal-to-Noise Ratio)

$$\frac{\sum_{i=1}^{Y} x^2(i)}{\sum_{i=1}^{Y} D^2(i)}$$

(Please refer to the table on the third page for the meanings of the parameters.)

Step 2 : **Mix clean speech and noise according to a specified signal-to-noise ratio (SNR) of 5 to generate noisy speech.**

Practice using `audio_SNR.m` by adding existing noise files such as train sounds, mouse sounds, etc., and adjusting the signal-to-noise ratio.

[output,fs]=audio_SNR('Clean filename.wav' , 'Noise filename.wav' ,SNR);

Experiment Two

Step 3 : Practice reading and playing audio files.

Use the commands `audioread` and `sound` to read and play clean speech and noisy speech.

[signal,fs]=audioread('filename.wav'); % Read in a WAV file (please make sure the WAV file is in the same directory).
sound(signal,fs); % Play the WAV file to listen.

Hint: Increasing the energy of the signal will make the sound louder. For example, `sound(signal*10, fs);`.

Step 4: Use the Wiener filter to generate noise-reduced signal (Enhancement Speech).

Pass the noisy speech signal through a Wiener filter to eliminate noise. Use `wienerscalart96.m` and input the signal variable `output`, sampling frequency `fs`, and the noise sample in seconds `s` (recommended between 0 and 1) to remove noise from the noisy speech signal.

output1=WienerScalart96(output,fs,s);
sound(output1,fs);

Step 5: Write a program to quantitatively evaluate the performance of the filter(compare the difference between the noise-reduced speech and the original clean speech).

SSNR Function:

$$segmental\ SNR(SSNR) = \frac{1}{m}\sum_{k=1}^{m}\frac{\sum_{i=1}^{N=180} x_k^2(i)}{\sum_{i=1}^{N=180}\left(x_k(i)-D_k(i)\right)^2}$$

The SSNR (Signal-to-Segmental Noise Ratio) of the clean speech after noise reduction

$$segmental\ SNR\ (SSNR) = \frac{1}{m}\sum_{k=1}^{m}\frac{\sum_{i=1}^{N=180} x_k^2(i)}{\sum_{i=1}^{N=180}\left(x_k(i)-D1_k(i)\right)^2}$$

The SSNR (Signal-to-Segmental Noise Ratio) of the noisy speech.

$$segmental\ SNR\ (SSNR) = \frac{1}{m}\sum_{k=1}^{m}\frac{\sum_{i=1}^{N=180} x_k^2(i)}{\sum_{i=1}^{N=180}\left(x_k(i)-D2_k(i)\right)^2}$$

The evaluation program is divided into two parts, each calculates its own SSNR. Finally, the SSNRs are combined using the formula in blue to calculate the SSNRI (Segmental Signal-to-Noise Ratio Improvement), which serves as the final evaluation criterion.
SSNRI (Segmental Signal-to-Noise Ratio Improvement) is calculated as the difference between the SSNR (Signal-to-Segmental Noise Ratio) of the clean speech after noise reduction and the SSNR of the noisy speech.

Variable Name	Meaning
X	Clean speech
D	Noise
D1	Clean speech after noise reduction (clean speech + noise, processed with a Wiener filter)

D2	Noisy speech (speech processed with `audio_SNR.m`)
N	Frame length (set to 180 for this project)
*Y	Audio length. For example, if an audio file has 1000 points, then in step one, you square and sum the 1000 points.
M	Number of frames = (Y/N), where any remainder is discarded. For example, $x_k(i)$ is the ith value in the kth frame of the clean speech x.

Hint1：When the program calculates SSNR, please pay attention to the dimensions and lengths of the two inputs. The lengths of clean speech (variable x) and processed speech (variable D) are different. Please use the length of the audio after Wiener filtering (variable D1) as the standard.

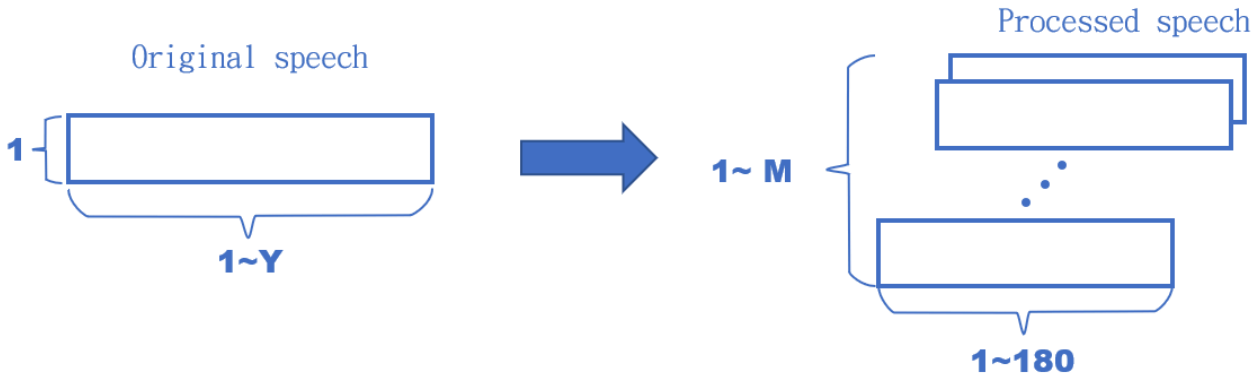
Hint2：For the audio files 1.wav to 10.wav which are stereo (dual-channel), please process the left and right channels sequentially during processing, and then take the average of the results from the two channels.

Hint3：The term "processed speech" can refer to either the part with added noise (noisy) or the enhanced part after processing (enhancement).

Hint4：Assuming you have written the program according to the definitions and methods provided above, if you use clean speech as '01', noise as 'horn', a signal-to-noise ratio of 10, mix them to create noisy speech, then use a Wiener filter with s=0.8, and in the SSNRI calculation program, set N (frame length) to 180, you will get *SSNRI=13.532.

*Note: The value of SSNRI may vary depending on the specific implementation and parameters used in the program.

*Additional Note: After splitting, the clean audio file X, originally of length Y, will result in M frames, each of length 180 (the final frame, if shorter than 180, will be discarded).



Experiment Three

Generate and play echo (please use the provided `echo.m` script).

In this experiment, we will use the `echo.m` tool to generate room echo. This script includes tools for Room Impulse Response Generator (RIRG). First, input the necessary parameters (room size, speaker position, microphone position, number of reflections, etc.), and RIRG will generate the Impulse Response (h) of the room. Then, convolve h with the input audio to generate the echo of the room.

```
[input,fs]=audoiread('filename');      % Read in the provided WAV file and its sampling frequency.
mic=[Length, width, height];           % Microphone position (in meters)
nn=                    % Number of reflections (please set between 1 and 5)
src=[Length, width, height];                  % Speaker position (in meters)
rm=[Length, width, height];            % Room size (in meters)
a=1                    % Reflection coefficient (please set between -1 and 1)
h=rir(fs, mic, nn, a, rm, src);        % Generate the room's impulse response (h).
d=filter(h,1,input);
sound(d,fs);                  % Play the echo.
```

Hint: Please make sure the WAV file is in the same directory.

Experiment Four

***Before proceeding with Experiment 4, please complete the echo generation in Experiment 3.**

Use the provided LMS program to perform echo cancellation (Matlab).

Use the provided `LMS.m` program to observe the estimated impulse response of the filter $\hat{h}$ compared to the simulated impulse response h from RIRG. You can use the `hold on` command and change colors to compare the two. The closer they are, the more accurate the estimated $\hat{h}$ of the filter.

For this tool, due to hardware limitations and for future experiment convenience, the parameters cannot be self-set. Please input the parameters according to the following table. Use Gaussian noise as the input audio.

Input	Randn(1,8e4)

Fs	10000
Mic (microphone position)	[1 2 2]
Src (source position)	[2 3 1]
Rm (room size)	[5 5 4]
a (reflection coefficient)	1
nn (number of reflections)	1
N (filter length)	291
mu (step size)	**0.0006**(adjustable)

The MSE is calculated as follows:

$$10e(n)^2$$

This calculation method serves two purposes in this experiment:

- Convert the units to dB.
- You can observe the detailed numerical values of the error as it converges to zero.

The MSE calculation program is already included in `LMS.m`, which calculates `mse_lms`. You just need to plot `mse_lms` to observe the convergence.

Experiment Five

Update NLMS(Normalized Least Mean Squares)

Regarding the echo cancellation filter, LMS is just one type.

Besides LMS, there are other filters with better performance. Therefore, in Experiment Four, we will

modify the algorithm in full_lms.m to update LMS to NLMS. We will compare the performance of the two filters in estimating $\hat{h}$ and MSE to see which one is better.

The following is the calculation formula for NLMS:

$$\hat{h}(n+1) = \hat{h}(n) + \frac{\mu[x(n)e(n)]}{[\alpha + x^T(n)x(n)]}$$

α: The small positive constant α = 1e-10

Compared to LMS:

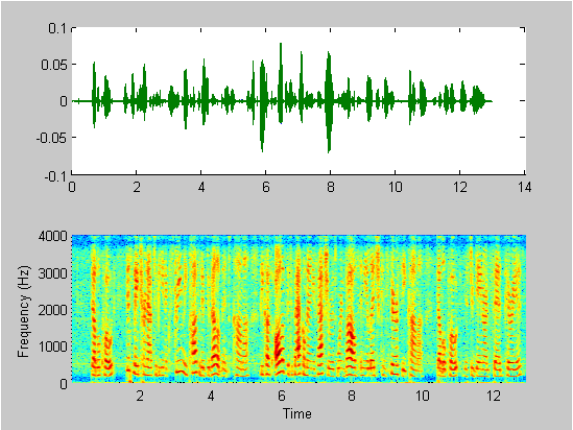
$$\hat{h}(n+1) = \hat{h}(n) + \mu[x(n)e(n)]$$

Hint: For detailed information, refer to the explanation in the experimental theory below.

IV. Supplementary Explanation

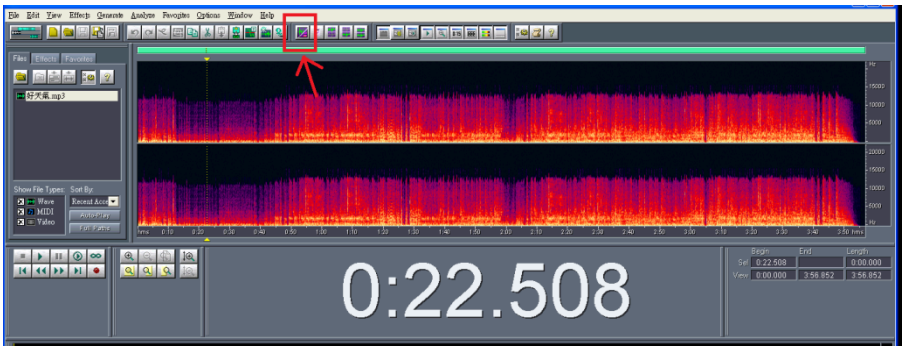
Plot the waveform and spectrogram using Matlab.

```
[signal,fs]=audioread('1.wav');
t=(0:length(signal(:,1))-1)/fs;
subplot(2,1,1);
plot(t,signal)
subplot(2,1,2);
spectrogram(signal(:,1), 1024, 1000, [], fs, 'yaxis')
```



Plot the spectrogram using Cool Edit.

You can download Cool Edit, an audio editing software that displays the spectrum of audio tracks. You can compare the spectrum graphs of both input and output. Simply import the audio file into Cool Edit and click on the icon to display the spectrum graph.



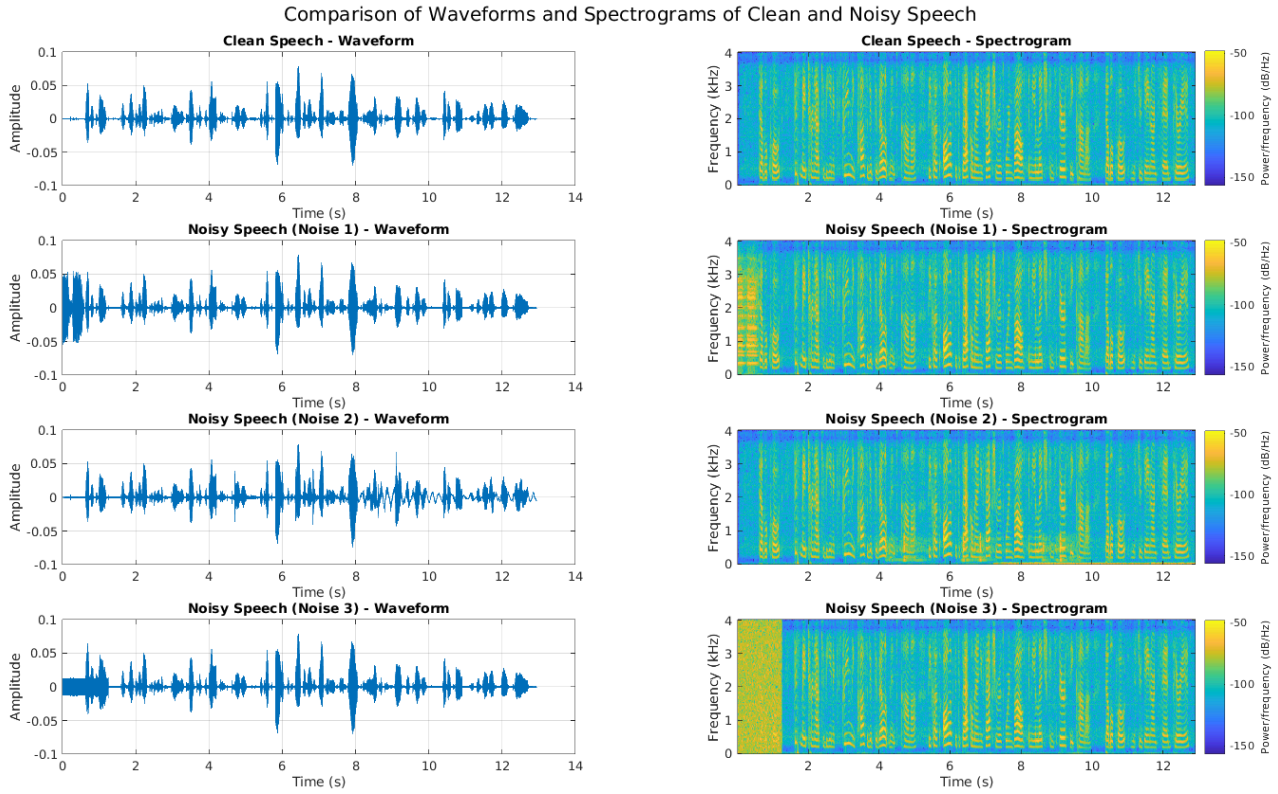
For Written Report

Question One:

1- Save the audio file after removing the noise using wienerscalart96.m (student ID) and upload it.

(Use the clean speech file 2.wav mixed with the horn noise, with a signal-to-noise ratio of 5)

2- Use clean speech 1.wav and three types of noise. Plot the waveform and spectrogram of the clean speech and noisy speech for each type of noise, and compare the differences.



Question Two:

Utilizing **Experiment Five,** write a short program to automate the process of quantizing 10 clean speech files against 3 different noise types (horn, mouse, and white noise), 2 signal-to-noise ratios (10, 50), noise sampling duration s = 0.5, N=180, for a total of 60 combinations. Create a table to present the data. **Please observe and compare the data, and provide an explanation.**

For example: When using clean speech '3', noise 'horn', a signal-to-noise ratio of 10, and a noise sampling rate of 0.8 seconds, the quantified value is *11.546.

N = The frame length is 180. Please refer to the SSNR theory section on page 2 for the concept of frames.

m = The number of frames = $(\frac{frame\ length}{N})$, if it does not divide evenly, discard the decimal point.

When calculating SSNR, ensure that both input variables, clean speech (variable x) and processed speech (variable D), have the same dimensions and length. **If their lengths are different, use the shorter length as the reference.**

DSPLAB Midterm project (1)

CleanSpeechFile	NoiseType	SNR	EnergyCleanSpeech	EnergyNoisySpeech
"1.wav"	"horn.wav"	10	3.6999e-05	4.0698e-05
"1.wav"	"horn.wav"	50	3.6999e-05	3.7e-05
"1.wav"	"mouse.wav"	10	3.6999e-05	4.0699e-05
"1.wav"	"mouse.wav"	50	3.6999e-05	3.7e-05
"1.wav"	"whitenoise.wav"	10	3.6999e-05	4.07e-05
"1.wav"	"whitenoise.wav"	50	3.6999e-05	3.7e-05
"2.wav"	"horn.wav"	10	2.7211e-05	2.9933e-05
"2.wav"	"horn.wav"	50	2.7211e-05	2.7211e-05
"2.wav"	"mouse.wav"	10	2.7211e-05	2.9934e-05
"2.wav"	"mouse.wav"	50	2.7211e-05	2.7211e-05
"2.wav"	"whitenoise.wav"	10	2.7211e-05	2.9931e-05
"2.wav"	"whitenoise.wav"	50	2.7211e-05	2.7211e-05
"3.wav"	"horn.wav"	10	0.00017096	0.00018805
"3.wav"	"horn.wav"	50	0.00017096	0.00017096
"3.wav"	"mouse.wav"	10	0.00017096	0.00018807
"3.wav"	"mouse.wav"	50	0.00017096	0.00017096
"3.wav"	"whitenoise.wav"	10	0.00017096	0.00018805
"3.wav"	"whitenoise.wav"	50	0.00017096	0.00017096
"4.wav"	"horn.wav"	10	0.00013939	0.00015333
"4.wav"	"horn.wav"	50	0.00013939	0.00013939
"4.wav"	"mouse.wav"	10	0.00013939	0.00015333
"4.wav"	"mouse.wav"	50	0.00013939	0.00013939
"4.wav"	"whitenoise.wav"	10	0.00013939	0.00015333
"4.wav"	"whitenoise.wav"	50	0.00013939	0.00013939
"5.wav"	"horn.wav"	10	0.00017128	0.00018842
"5.wav"	"horn.wav"	50	0.00017128	0.00017129
"5.wav"	"mouse.wav"	10	0.00017128	0.00018842
"5.wav"	"mouse.wav"	50	0.00017128	0.00017129
"5.wav"	"whitenoise.wav"	10	0.00017128	0.00018843
"5.wav"	"whitenoise.wav"	50	0.00017128	0.00017129
"6.wav"	"horn.wav"	10	0.00041698	0.00045867
"6.wav"	"horn.wav"	50	0.00041698	0.00041699
"6.wav"	"mouse.wav"	10	0.00041698	0.00045868
"6.wav"	"mouse.wav"	50	0.00041698	0.00041699
"6.wav"	"whitenoise.wav"	10	0.00041698	0.00045868
"6.wav"	"whitenoise.wav"	50	0.00041698	0.00041699
"7.wav"	"horn.wav"	10	0.00025212	0.00027737
"7.wav"	"horn.wav"	50	0.00025212	0.00025213
"7.wav"	"mouse.wav"	10	0.00025212	0.00027733
"7.wav"	"mouse.wav"	50	0.00025212	0.00025213
"7.wav"	"whitenoise.wav"	10	0.00025212	0.0002773
"7.wav"	"whitenoise.wav"	50	0.00025212	0.00025213
"8.wav"	"horn.wav"	10	7.4975e-05	8.2473e-05
"8.wav"	"horn.wav"	50	7.4975e-05	7.4976e-05
"8.wav"	"mouse.wav"	10	7.4975e-05	8.2471e-05
"8.wav"	"mouse.wav"	50	7.4975e-05	7.4976e-05
"8.wav"	"whitenoise.wav"	10	7.4975e-05	8.2471e-05
"8.wav"	"whitenoise.wav"	50	7.4975e-05	7.4976e-05
"9.wav"	"horn.wav"	10	0.00021693	0.00023862
"9.wav"	"horn.wav"	50	0.00021693	0.00021693
"9.wav"	"mouse.wav"	10	0.00021693	0.00023863
"9.wav"	"mouse.wav"	50	0.00021693	0.00021693
"9.wav"	"whitenoise.wav"	10	0.00021693	0.00023861
"9.wav"	"whitenoise.wav"	50	0.00021693	0.00021693
"10.wav"	"horn.wav"	10	9.7644e-05	0.0001074
"10.wav"	"horn.wav"	50	9.7644e-05	9.7644e-05
"10.wav"	"mouse.wav"	10	9.7644e-05	0.0001074
"10.wav"	"mouse.wav"	50	9.7644e-05	9.7644e-05
"10.wav"	"whitenoise.wav"	10	9.7644e-05	0.00010741
"10.wav"	"whitenoise.wav"	50	9.7644e-05	9.7645e-05

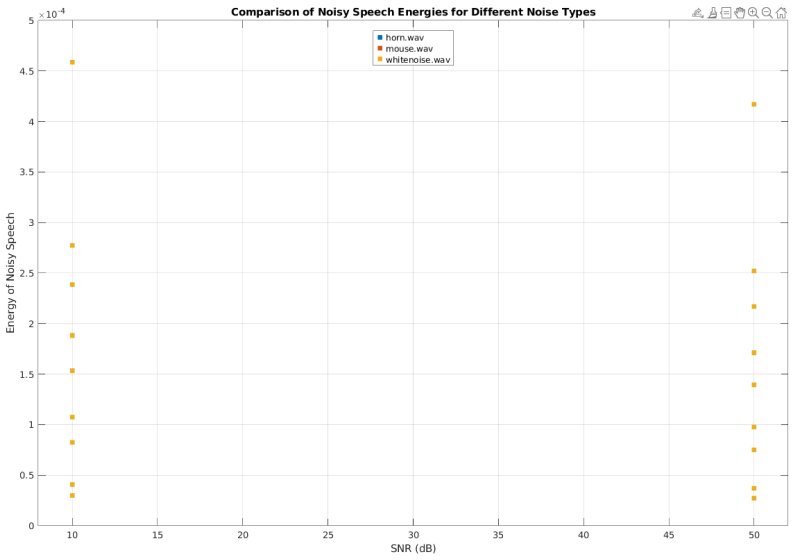


Table example: :

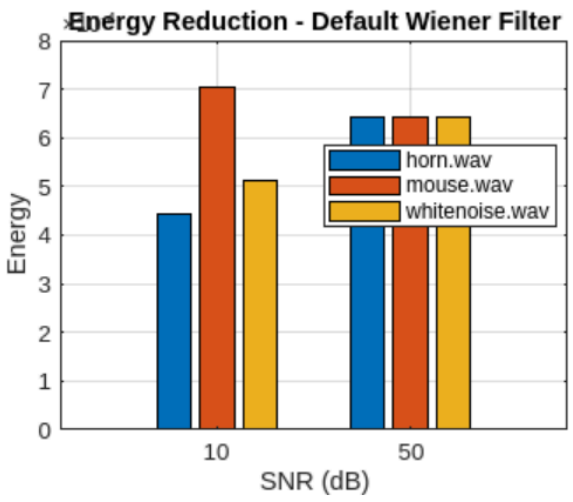
Table 1: Clean Speech: '03'

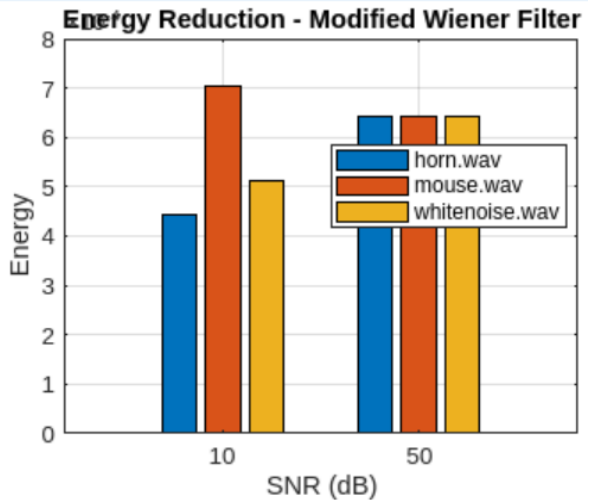
SNR Noise	10	50
horn	*11.546	
mouse		

Question Three :

Adjust the parameters *IS, W, and SP in wienerscalart96.m to improve the performance of the Wiener Filter in noise reduction. Create the same table as in **Question Two**, and observe and compare the quantified values before and after modifying the parameters in wienerscalart96.m.

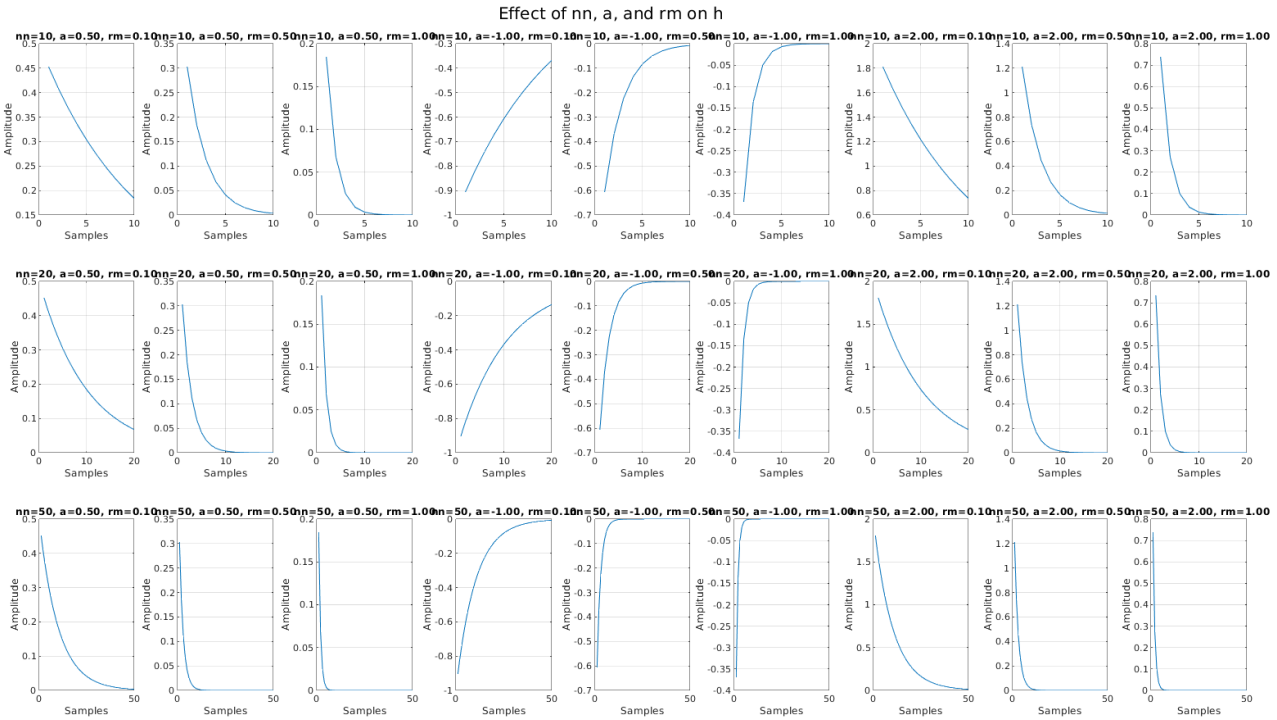
CleanSpeechFile	NoiseType	SNR	EnergyNoisySpeech	EnergyDenoisedDefault	EnergyDenoisedModified
"1.wav"	"horn.wav"	10	4.0698e-05	9.9451e-05	9.9451e-05
"1.wav"	"horn.wav"	50	3.7e-05	0.00014761	0.00014761
"1.wav"	"mouse.wav"	10	4.0633e-05	0.00016216	0.00016216
"1.wav"	"mouse.wav"	50	3.6999e-05	0.00014766	0.00014766
"1.wav"	"whitenoise.wav"	10	4.0711e-05	0.00011651	0.00011651
"1.wav"	"whitenoise.wav"	50	3.7e-05	0.00014764	0.00014764
"2.wav"	"horn.wav"	10	2.9933e-05	7.2507e-05	7.2507e-05
"2.wav"	"horn.wav"	50	2.7211e-05	0.00010854	0.00010854
"2.wav"	"mouse.wav"	10	2.9816e-05	0.00011897	0.00011897
"2.wav"	"mouse.wav"	50	2.721e-05	0.00010857	0.00010857
"2.wav"	"whitenoise.wav"	10	2.9828e-05	8.4911e-05	8.4911e-05
"2.wav"	"whitenoise.wav"	50	2.721e-05	0.00010856	0.00010856
"3.wav"	"horn.wav"	10	0.00018805	0.00043161	0.00043161
"3.wav"	"horn.wav"	50	0.00017096	0.00068291	0.00068291
"3.wav"	"mouse.wav"	10	0.0001881	0.00075161	0.00075161
"3.wav"	"mouse.wav"	50	0.00017096	0.00068315	0.00068315
"3.wav"	"whitenoise.wav"	10	0.00018793	0.00052125	0.00052125
"3.wav"	"whitenoise.wav"	50	0.00017096	0.00068305	0.00068305
"4.wav"	"horn.wav"	10	0.00015333	0.00037424	0.00037424
"4.wav"	"horn.wav"	50	0.00013939	0.00055704	0.00055704
"4.wav"	"mouse.wav"	10	0.00015337	0.00061307	0.00061307
"4.wav"	"mouse.wav"	50	0.00013939	0.00055724	0.00055724
"4.wav"	"whitenoise.wav"	10	0.00015317	0.00043367	0.00043367
"4.wav"	"whitenoise.wav"	50	0.00013939	0.00055715	0.00055715
"5.wav"	"horn.wav"	10	0.00018842	0.00045114	0.00045114
"5.wav"	"horn.wav"	50	0.00017129	0.00068403	0.00068403
"5.wav"	"mouse.wav"	10	0.00018839	0.00075253	0.00075253
"5.wav"	"mouse.wav"	50	0.00017129	0.00068424	0.00068424
"5.wav"	"whitenoise.wav"	10	0.00018866	0.0005306	0.0005306
"5.wav"	"whitenoise.wav"	50	0.00017129	0.00068416	0.00068416
"6.wav"	"horn.wav"	10	0.00045867	0.0012208	0.0012208
"6.wav"	"horn.wav"	50	0.00041699	0.0016672	0.0016672
"6.wav"	"mouse.wav"	10	0.0004579	0.0018312	0.0018312
"6.wav"	"mouse.wav"	50	0.00041698	0.0016677	0.0016677
"6.wav"	"whitenoise.wav"	10	0.000458	0.0013729	0.0013729
"6.wav"	"whitenoise.wav"	50	0.00041698	0.0016675	0.0016675
"7.wav"	"horn.wav"	10	0.00027737	0.00069065	0.00069065
"7.wav"	"horn.wav"	50	0.00025213	0.0010071	0.0010071
"7.wav"	"mouse.wav"	10	0.00027687	0.0011062	0.0011062
"7.wav"	"mouse.wav"	50	0.00025212	0.0010074	0.0010074
"7.wav"	"whitenoise.wav"	10	0.00027746	0.00080941	0.00080941
"7.wav"	"whitenoise.wav"	50	0.00025213	0.0010073	0.0010073
"8.wav"	"horn.wav"	10	8.2473e-05	0.00022262	0.00022262
"8.wav"	"horn.wav"	50	7.4976e-05	0.00029933	0.00029933
"8.wav"	"mouse.wav"	10	8.2651e-05	0.00033007	0.00033007
"8.wav"	"mouse.wav"	50	7.4977e-05	0.00029944	0.00029944
"8.wav"	"whitenoise.wav"	10	8.2492e-05	0.00024751	0.00024751
"8.wav"	"whitenoise.wav"	50	7.4976e-05	0.00029939	0.00029939
"9.wav"	"horn.wav"	10	0.00023862	0.00060952	0.00060952
"9.wav"	"horn.wav"	50	0.00021693	0.00086653	0.00086653
"9.wav"	"mouse.wav"	10	0.00023856	0.0009532	0.0009532
"9.wav"	"mouse.wav"	50	0.00021693	0.00086683	0.00086683
"9.wav"	"whitenoise.wav"	10	0.00023858	0.00069697	0.00069697
"9.wav"	"whitenoise.wav"	50	0.00021693	0.0008667	0.0008667
"10.wav"	"horn.wav"	10	0.0001074	0.00027963	0.00027963
"10.wav"	"horn.wav"	50	9.7644e-05	0.00039005	0.00039005
"10.wav"	"mouse.wav"	10	0.00010742	0.00042921	0.00042921
"10.wav"	"mouse.wav"	50	9.7645e-05	0.00039017	0.00039017
"10.wav"	"whitenoise.wav"	10	0.00010728	0.00032065	0.00032065
"10.wav"	"whitenoise.wav"	50	9.7643e-05	0.00039011	0.00039011





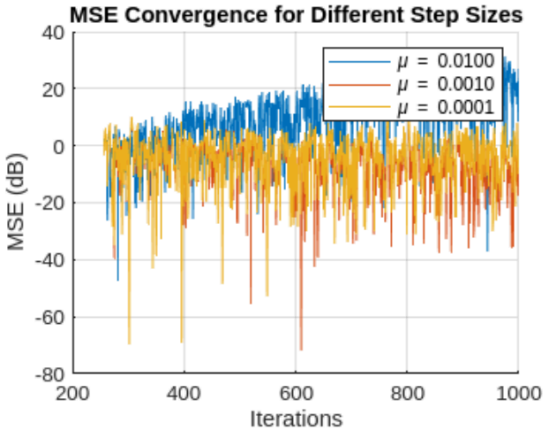
Question Four :

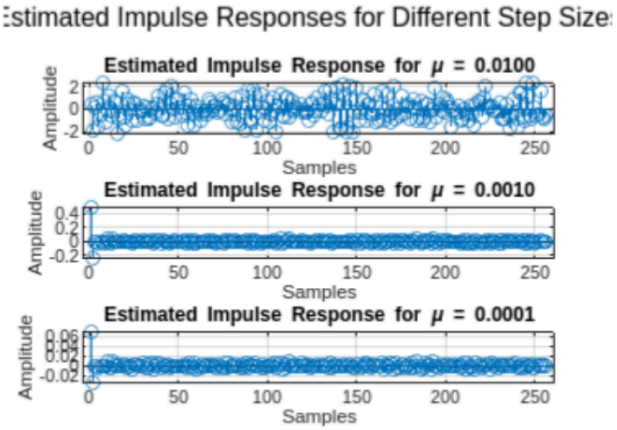
For each parameter nn, a, and rm in Experiment 3, input different values and plot h. Describe the changes and the perceived echo for at least three different combinations. (e.g., How does increasing nn affect the result? What happens if a is set to -1?)A



Question Five :

Input different values of μ (step size) to observe the convergence of MSE and the estimated impulse response $\hat{h}$. Compare the MSE for at least three different values of μ (large, medium, small), and explain whether increasing or decreasing μ affects the convergence speed of MSE. What happens if μ is too large? Too small?





Effect of Large μ:

- Faster updates but can cause oscillations or divergence if too large.
- In extreme cases, μ>1/power of input signal\mu > 1/\text{power of input signal}μ>1/power of input signal leads to instability.

Effect of Small μ:

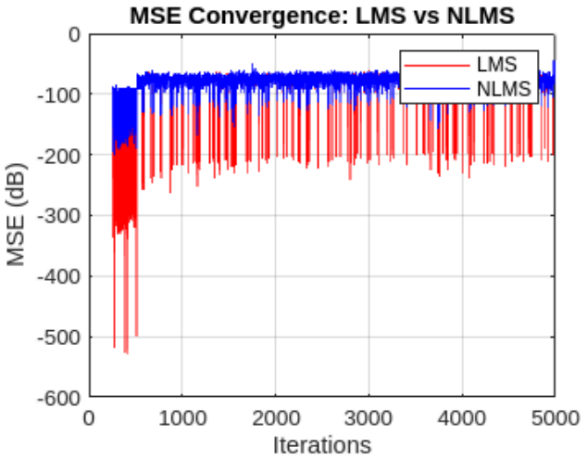
- Updates are slow, requiring more iterations for convergence.
- May not fully converge within a limited number of iterations.

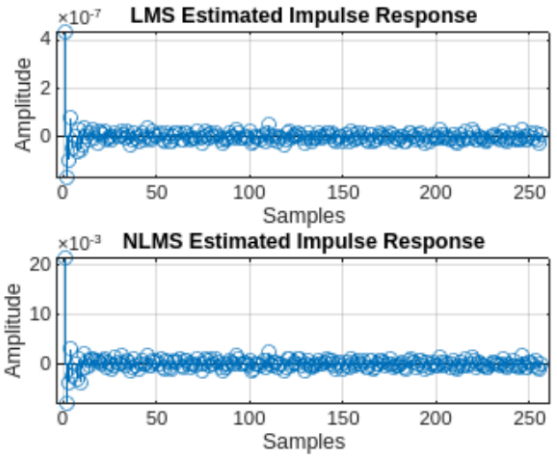
Optimal μ:

- A moderate value balances convergence speed and stability.
- Choose μ\muμ based on the signal characteristics and desired performance.

Question Six :

Compare the differences between LMS and NLMS (convergence of MSE, performance when input is changed to speech).





C. Scoring Criteria

Written Report (80%)

Please provide detailed answers to the six questions and discussions, and compile them into a report. Upload the report and MATLAB files to the portal before December 10th, and submit a printed copy in class on that day.

The report accounts for 80% of your grade, with each experiment contributing 10% and each question discussion contributing 5%.

MATLAB Files & Audio Files (20%)

Please upload the code to the portal by December 10th and the audio file for **Question One** on November 26th.

D. Appendix

Wiener Filter

The Wiener filter, proposed by Norbert Wiener in 1949, is a type of linear filter also known as a Minimum Mean Square Error (MMSE) filter or a Least Square Error filter, as it aims to minimize the error. The Wiener filter not only removes noise but also finds applications in various fields such as single-source signal separation, image restoration, and image noise reduction. In this context, we employ the Wiener filter to filter the speech signal received by a microphone.

Let's assume that y[n] represents a signal contaminated with noise, x[n] denotes a clean original speech signal, and d[n] represents noise or interference. This relationship can be expressed by the following equation.

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

The objective of filtering through the filter is to remove the noise d[n] from the signal y[n] so that we obtain an approximation, x̂[n], of the clean original speech signal x[n]. Assuming this to be a linear filter, its design method is based on statistical results, where appropriate selection of filter coefficients minimizes the difference between the filtered signal x̂[n] and the original speech signal x[n]. Let's denote the impulse response of this filter as h[n], where the input is the signal y[n] contaminated with noise, and the output is x̂[n], an approximation of x[n]. This relationship can be expressed by the following equation.

$$\hat{x}[n] = \sum_{k=0}^{\infty} y[n-k]h[k] \tag{2}$$

The computed error energy can be expressed as Equation (3).。

$$E(d) = E\left\{\left(x[n] - \hat{x}[n]\right)^2\right\} \tag{3}$$

Expanding the above equation yields Equation (4).

$$E(d) = \sum_{n=0}^{\infty}\left\{x[n] - \sum_{k=0}^{\infty} y[n-k]h[k]\right\} \tag{4}$$

E(d) represents the computed energy error, estimating the squared error between x[n] and x̂[n].

To select an impulse response h[n] that minimizes the error, differentiating with respect to h[n] in the

above equation yields Equation (5). Expanding Equation (5) further gives us Equation (6).

$$\sum_{n=0}^{\infty}\left[\left(x[n] - \sum_{k=0}^{\infty} y[n-k]h[k]\right)y[n-i]\right] = 0 \tag{5}$$

$$\sum_{n=0}^{\infty}(x[n]y[n-i]) - \sum_{k=0}^{\infty} h[k]\sum_{n=0}^{\infty}(y(n-k)y[n-i]) = 0 \tag{6}$$

From Equation (6), we can derive the following two equations, (7) and (8).

$$\sum_{n=0}^{\infty}(x[n]y[n-i]) = \sum_{n=0}^{\infty}[x[n](x[n-i] + d[n-i])]$$

$$= \sum_{n=0}^{\infty} x[n](x[n-i] + x[n]d[n-i]) \tag{7}$$

And

$$\sum_{n=0}^{\infty}(y[n-k]y[n-i]) = \sum_{n=0}^{\infty}[x[n](x[n-i]+d[n-i])]$$
$$= \sum_{n=0}^{\infty}(x[n-k]+d[n-k])(x[n-i]+d[n-i]) \quad (8)$$

Assuming that d[n] and x[n] are uncorrelated, Equation (9) holds.

$$\sum_{n=0}^{\infty}(x[n]d[n-i]) = 0 \quad (9)$$

Therefore, we can deduce Equation (10) and Equation (11). These two equations are derived to simplify Equation (6) into the form of r_x, where each part is represented by r_x[i] and r_x[i-k]+r_d[i-k]. This simplification makes it more concise when transforming to the frequency domain and helps to clearly understand the performance of the filter.

$$\sum_{n=0}^{\infty}(x[n]y[n-i]) = \sum_{n=0}^{\infty}[x[n](x[n-i])+d[n-i])]$$
$$= \sum_{n=0}^{\infty}(x[n]x[n-i]+x[n]d[n-i])$$
$$= \sum_{n=0}^{\infty}x[n]x[n-i]$$
$$= r_x[i] \quad (10)$$

$$\sum_{n=0}^{\infty}(y[n-k]y[n-i])$$
$$= \sum_{n=0}^{\infty}(x[n-k]+d[n-k])(x[n-i]d[n-i])$$
$$= \sum_{n=0}^{\infty}x[n-k]x[n-i]+d[n-k]d[n-i]$$
$$= r_x[i-k]+r_d[i-k] \quad (11)$$

Substituting Equation (11) into Equation (6), we obtain Equation (12).

$$r_x[i] = \sum_{k=0}^{\infty}h[k]\left(r_x[i-k]+r_d[i-k]\right) \quad (12)$$

Transferring Equation (12) to the frequency domain yields Equation (13).

$$f_x\left(e^{jw}\right) = H\left(e^{jw}\right)\left[f_x\left(e^{jw}\right)+f_d\left(e^{jw}\right)\right] \quad (13)$$

Translating h[n] from the time domain to the frequency domain and then rearranging terms, we obtain$H\left(e^{jw}\right)$。

$$H\left(e^{jw}\right) = \frac{f_x\left(e^{jw}\right)}{f_x\left(e^{jw}\right)+f_d\left(e^{jw}\right)} \quad (14)$$

This is the form of the Wiener filter.Where $f_i\left(e^{jw}\right)$ represents the power spectral density (PSD) of

signal i. By Fourier transforming the signal with noise $f_y(e^{jw})$, we can obtain the Fourier transform $\hat{f}x(e^{jw})$ of $\hat{x}[n]$

$$\hat{f}x(e^{jw}) = H(e^{jw})f_y(e^{jw}) \tag{15}$$

In the derivation above, the Wiener filter is established under the assumption that the noise d[n] is uncorrelated with the clean original speech signal x[n]. However, in practical applications, computations need to be performed within short time frames,typically within one audio frame. Hence, the windowed audio frame signal is represented as Equation (15).

$$f_y(m;n) = f_s(m;n) + f_d(m;n) \tag{16}$$

m represents the position of the audio frame. Writing a Wiener filter within a single audio frame is represented by Equation (17).

$$H(e^{jw};m) = \frac{\hat{f}_x(e^{jw};m)}{\hat{f}_x(e^{jw};m)+\hat{f}_d(e^{jw};m)} \tag{17}$$

The signals and noise can only be estimated and their exact values are not known. Therefore, their power spectral densities (PSD) are represented as estimates. Passing an audio frame signal \(f_y(m;n) \) through a Wiener filter in the frequency domain yields the estimated speech signal.

Our objective here is noise reduction, i.e., using a Wiener filter to eliminate noise that might interfere with microphone speech recognition. Equation (14) can be utilized to represent this filter. Leveraging the concept of noise separation using Equation (14), let S_1 denote the desired speech signal to be preserved, and S_2 represent the noise signal. S_1 andS_2 are defined as follows. The power spectral density in these equations is expressed as: The power spectral density is then represented by$f_x(e^{jw})$.

$$S_1 = \sum_{k\in K_1} a_k b_k$$

$$S_2 = \sum_{k\in K_2} a_k b_k \tag{18}$$

The equation (18) above expresses the signalsS_1 、S_2 It can be viewed as composed of linear combinations of b_k .where b_k represents the components forming the signal and can also be seen as the features of the signal，K_1 、K_2 respectively denote the feature sets of the speech signal and the noise signal, while a_k represents the weights of each feature. Finally, by multiplying each signal component b_k with its corresponding weight a_k ，we obtain the results of their respective linear combinationsS_1 、S_2。Since we're utilizing the Wiener filter, we represent features using power spectral density. Transitioning from Equation (18) to the frequency domain yields Equation (19).

$$F\{S_1(f,t)\} = \sum_{k\in K_1} a_{k(t)} \times F\{b_k(f)\}$$

$$F\{S_2(f,t)\} = \sum_{k\in K_2} a_{k(t)} \times F\{b_k(f)\} \tag{19}$$

The frame position index is denoted as (t). With this, using the Wiener filter to separate the noisy part from the speech signal, F{x(f,t)} can be expressed as the following equation.

$$F\{\hat{S}_1(f,t)\} = H_1 \times F\{x(f,t)\}$$
$$F\{\hat{S}_2(f,t)\} = H_2 \times F\{x(f,t)\} \tag{20}$$

Where H_1、H_2 can be represented as Equations (21) and (22), respectively.

$$H_1 = \frac{\sum_{k\in K_1} a_{k(t)} \times F\{b_k(f)\}}{\sum_{k\in K_1 \cup K_2} a_{k(t)} \times F\{b_k(f)\}} \tag{21}$$

$$H_1 = \frac{\sum_{k\in K_2} a_{k(t)} \times F\{b_k(f)\}}{\sum_{k\in K_1 \cup K_2} a_{k(t)} \times F\{b_k(f)\}} \tag{22}$$

Where the speech signal contaminated with noise can be represented as Equation (23).

$$F\{x(f,t)\} \approx \sum_{k\in K_1 \cup K_2} a_{k(t)} \times F\{b_k(f,t)\} \tag{23}$$

Based on the above description and derivation, we can see that if a Wiener filter is used to remove noise from a speech signal, the required elements are S_1 for the speech signal to be preserved; S_2 the noise signal; their respective features b_k; and the feature weights a_k From equation (23), it is evident that〖 a〗_kand b_k are the key variables to be determined. The process of removing noise from a speech signal can be divided into two parts: a training process and a separation process.During the training process, the objective is to extract features from both speech and noise signals and store them in a database. Since the Wiener filter utilizes Power Spectral Density (PSD) to separate signals, PSD serves as the desired feature in this context. During the separation process, the features obtained from the training process are used along with the Wiener filter concept to separate the speech information containing noise.

In the training process, the speech signals containing noise are input, preprocessed, and then the Power Spectral Density is calculated. The features extracted during training are then utilized to separate the speech and noise signals, each representing different features. Preprocessing typically involves Fourier Transform, resampling, amplitude normalization, and framing.

SNR (Signal-to-noise ratio)

There are several objective speech quality measures. The most simple one is the Signal to Noise Ratio (SNR) that compares the original and processed speech signals sample by sample. There are also more complex ones that are built based on Human Auditory System model involving complex

mathematical calculations. We present the most famous measures in this section. All of them with the exception of the ITU E-model operates on both the original and the processed speech sample. This limitation makes it impossible to work in real time and to include these metrics in designing new mechanisms (rate control or speech codecs design to take into account the user's perception and the network factors). A second disadvantage is that the obtained results do not correlate always with subjective data (thus they cannot measure correctly user's perception). A third drawback is that some of them are computationally extensive. This point limits their usage in lightweight applications including mobile phones. Some of these metrics are designed and optimized basically to consider encoding impairments and restricted conditions, but they do not work efficiently when they used in other conditions (ex. distortion due to the transmission over the network). Some of these methods require a perfect synchronization between the original and processed signals otherwise the performce degrades considerably. In this case several factors including the delay variation's effect cannot be taken into account by these methods. There are three types of objective speech quality measures: time domain, spectral domain, and perceptual domain measures. The time domain measures are usually applicable to analog or waveform coding systems in which the goal is to reproduce the waveform itself. SNR and segmental SNR (SNRseg) are the most known methods. Since the waveform are directly compared in time domain, synchronization of the original and distorted signals is a must. However, synchronization is difficult; if not performed well, the performance is poor. The most simple possible measure is the Signal-to-Noise (SNR) ratio. Its goal is to measure the distortion of the waveform coders that reproduce the input waveform. It is calculated as follows:

$$SNR = \frac{\sum_{i=1}^{N} x^2(i)}{\sum_{i=1}^{N} (x(i)-D(i))^2}$$

where $x(i)$ and $D(i)$ are the original and processed speech samples indexed by i and N is the total number of samples point.

Segmental Signal-to-Noise Ratio (segSNR), instead of working on the whole signal, calculates the average of the SNR values of short segments (15 to 20 ms). It is given by:

$$segSNR(SSNR) = \frac{1}{m} \sum_{k=1}^{m} \frac{\sum_{i=1}^{N} x_k^2(i)}{\sum_{i=1}^{N} \left(x_k(i)-D_k(i)\right)^2}$$

where N and m are the segment length and the number of segments respectively. segSNR gives better results than SNR for waveform encoders, but it gives very bad results for vocoders. The second type of measures are the spectral domain ones [1]. They are generally computed using speech segments typically between 15 and 30 ms long. They are much more reliable than time domain measures and less sensitive to the misalignments between the original and distorted signals. However, these measures are closely related to speech codec design and use the parameters of speech production modules. Hence their ability to adequately describe the listener's auditory response is limited by the constraints of the speech production modules. They include the log likelihood ratio, the Linear Predictive Coding (LPC) parameter distance measures, the cepstral distance, and the weighted slope spectral distance measures (for more details and descriptions see [1]). In general, all these methods gives good results for some encoding distortion, but they are

not valid for the case when the original speech is passed through a communication system that significantly changes the statistics of the original speech. The third type of objective measures is constituted by the perceptual domain measures [1]. In contrast to the spectral domain measures, perceptual domain measures are based on models of human auditory perception. They transform speech signal into a perceptually relevant domain such as bark spectrum or loudness domain, and incorporate human auditory models. They give better prediction of the quality under the condition that the used auditory model used truly describes the human auditorial system. It is clear that this task is very complex and it is not possible to implement exact model of such system. However, by using approximations of the human auditorial system, the obtained results correlate better than that of the other two types of speech measures. Another important point to underline is the fact that these models are optimized for a specific type of speech data; the performance is not good for different speech data. In addition, they have the risk of not describing perceptually important effects relevant to speech quality but simply a curve fitting by parameter optimization. These measures are the most known and used in the literature. We provide a brief description of these metrics as given in [1].

[1] W. Yang. "*Enhanced Modified Bark Spectral Distortion (EMBSD): an Objective Speech Quality Measrure Based on Audible Distortion and Cognition Model.*" PhD thesis, Temple University Graduate Board, May 1999.

Least mean square algorithm (LMS)

The Least Mean Squares (LMS) algorithm is the most widely used filtering algorithm, and its biggest

feature is its simplicity. The computation process involves only addition and multiplication, and it does not require calculation of the correlation function, nor does it need complex matrix inversion operations. As a result, it is often used as a benchmark for comparison.

The LMS algorithm minimizes the mean squared value of the output error signal,e^2最小化 which is defined as:

$$J(N) = e^2,\ n = 0,\ 1,\ 2\ldots$$

The difference between the estimated value of the target signal and the actual target signal is referred

to as the estimation error. The definition of e(n) is as follows:

$$e(n) = d(n) - \hat{h}^T(n)u(n)$$

In this context, $d(n)d(n)$ represents the desired value. $w(n)w(n)$ is the weight coefficient vector at time nn, and its expanded form is:

$$h(n) = \left[h_0\ h_1\ \ldots\ h_{L-1}\right]^T$$

$u(n)u(n)$ is the input vector, and its expanded form is:

$$x(n) = [x(n)\ x(n-1)...x(n-L+1)]^T$$

In this context, *LL* represents the length of the filter. The LMS algorithm focuses on adjusting the difference $e(n)$*e*(*n*) between the system's desired value $d(n)$*d*(*n*) and the filter output $y(n)$*y*(*n*). The algorithm continuously updates and modifies the values of the weight coefficient vector $h(n)$*h*(*n*) during computation, aiming to minimize the square of $e(n)$*e*(*n*), thereby reducing the error towards zero.

The description of each parameter is as follows:

n: The current time index

x(n): The vector of buffered input samples at step *n*

h(n): The vector of filter weight estimates at step *n*

y(n): The filtered output at step *n*

e(n): The estimation error at step *n*

d(n): The desired response at step *n*

μ: (The adaptation step size)

The following is the formula for the LMS algorithm (parameters are annotated with their dimensions, assuming a filter length *LL* of 400).

- Filter Output: $y(n) = \hat{h}^T(n-1)x(n)$

  1x1 1x400 400x1

- Estimation Error: $e(n) = d(n) - y(n)$

  1x1 1x1 1x1

- Tap-weight adaptation: $\hat{h}(n+1) = \hat{h}(n) + \mu[u(n)e(n)]$

  400x1 400x1

The selection of the step size parameter $\mu\mu$ in the LMS algorithm is indeed crucial. The value of $\mu\mu$ is used to adjust the correction speed of the weighted parameters. If $\mu\mu$ is chosen too small, the convergence speed may be excessively slow. Conversely, selecting a value that is too large can lead t

o unstable convergence, causing divergence. Therefore, finding the optimal $\mu\mu$ value is a significant challenge in the LMS algorithm. There are certain limitations on the selection of $\mu\mu$, and the

convergence condition is: $0 < \mu < \sum_{K=0}^{n} E\{|x(n-k)|^2\}$

Normalized LMS algorithm (NLMS):

NLMS algorithm redefines the μ value in LMS, allowing it to change with the normalization of the input signal, thereby improving the stability of convergence.

The following is the formula required for the NLMS algorithm:

$$\hat{h}(n+1) = \hat{h}(n) + \frac{\mu[x(n)e(n)]}{\left[\alpha + x^T(n)x(n)\right]}$$

α : The small positive constant α = 1e-10

Compared to LMS:

$$\hat{h}(n+1) = \hat{h}(n) + \mu[x(n)e(n)]$$

The definitions of the parameters are the same as in the LMS (Least Mean Squares) algorithm, but additional meanings for these parameters are introduced as follows:

The stability and convergence speed of the LMS algorithm are influenced by the value of μ and the reference signal. Since μ is a constant value, the overall convergence speed of LMS is affected by it, making its response to rapidly changing signals less than ideal. In contrast, the NLMS (Normalized Least Mean Squares) algorithm improves by addressing the impact of the input signal on the convergence factor. Here, μ changes over time, denoted as μ(n), allowing it to adjust dynamically to optimal values. Furthermore, to prevent divergence of the convergence factor (μ) when the input signal is too small, an α value is also incorporated.