Delft University of Technology

Estimation and Detection
ET4386

Multi-Microphone Speech Enhancement

Authors:
Yanqi Hong (5884683) y.hong-8@student.tudelft.nl
Kaishen Lin (5971950) K.Lin-11@student.tudelft.nl

January 7, 2024

1 Introduction

Multi-Microphone Speech Enhancement is a crucial technology in audio processing, particularly for applications in noisy environments such as mobile phones, hearing aids, and speech recognition systems. The technique employs multiple microphones at different locations to enhance speech signals. This spatial arrangement allows for better differentiation between the desired speech and background noise, a challenge for single-microphone systems. By mathematically modeling the signals from various microphones, these systems effectively isolate and enhance speech, improving clarity and intelligibility. This technology is increasingly vital in our noise-filled world, enhancing the user experience in everyday applications and specialized devices.

Multi-channel speech enhancement is a complex problem that requires many factors to be considered, such as delay and damping. However, in this problem, we assume that a key factor acoustic transfer function(ATF) of the target source always equals 1. It's an idealized scenario where the target sound source is located in the far-field of the microphone array where the sound waves can be considered parallel (with minimal divergence). Meanwhile, the target source is directly facing the array, resulting in the same sound delay and damping at all microphones. This greatly simplifies the process of processing speech signals. In our project, we first divide the signal into frames using a 50% overlap, 20 ms duration and apply a Hann window to each frame, then perform a Fast Fourier Transform (FFT) on each frame. Then we use the time frames during the first second, as these are noise only, and apply Voice Activity Detection (VAD) to better extract noise characteristics.Finally, We use MLE when assuming the speech signal is deterministic and MMSE when assuming the speech signal is stochastic.

2 Signal Model and assessment metrics

2.1 Signal Model

According to the description of our project, A noisy signal can be modeled as

$$Y_m(n) = S_m(n) + W_m(n) \tag{1}$$

where $S_m(n)$ is the target speech signal at sample-time index n and microphone m $\in \{1, 2, \ldots, M\}$ and $W_m(n)$ is the noise signal. After the framing and windowing, we do frequency transformation on each signal, then the signal model in the frequency domain is given by

$$Y_m(l,k) = A_m(l,k)S(l,k) + W_m(l,k) \tag{2}$$

where $S(l,k)$ is now the target source at the source location, and k indicates the frequency-bin index. we stack the DFT coefficients for the different microphones in a vector and assume that $A_m(l,k) = 1 \forall k, l, m$, we obtain

$$\mathbf{Y}(l,k) = \mathbf{S}(l,k) + \mathbf{W}(l,k) \tag{3}$$

The goal is to estimate the target speech signal in Python by inputting the `.csv` file converted from the given Matlab data. In the preprocessing of the data, we first divide the speech signal into frames using a 50% overlap, 20 ms duration and apply a Hann window to each frame, then perform an FFT on each frame. After FFT the spectrum of clean speech signal is shown in Figure 1, and the spectrum of channel 1 is shown in Figure 2. We obtain the spectrum of signal in a dB scale to make noise signals more prominent and visible.

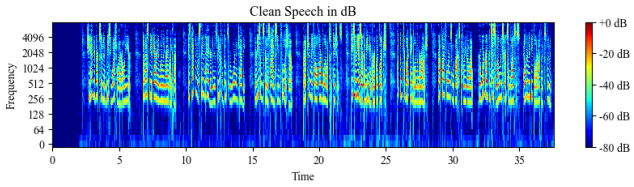

Figure 1: Clean speech spectrum

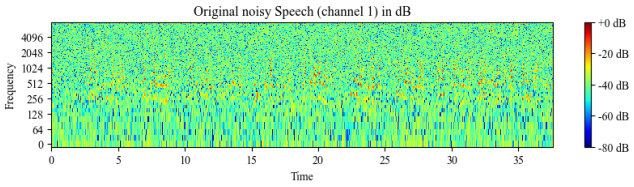

Figure 2: Noisy speech spectrum (channel 1)

2.2 Assessment metrics

In this project, we use three assessment metrics to evaluate the performance of our estimator.

2.2.1 Signal-to-Noise Ratio: SNR

SNR is a straightforward way to quantify the amount of noise reduction achieved by a speech enhancement model. Also, there is often a correlation between SNR and the perceived quality of speech. Higher SNR generally suggests clearer speech with less intrusive noise, which typically aligns with a better subjective listening experience. Equation 4 calculates SNR (γ_k) using $\sigma^2_{\mathrm{Y}_k}$ and $\sigma^2_{\mathrm{W}_k}$ for noisy speech and noise variances, respectively. As variance in a zero-mean signal signifies energy, γ_k represents the ratio of speech energy to noise energy. The noise energy is determined from the first-second data in the noisy speech, where no speech signal is present in the clean speech.

$$\gamma_k = 10\log_{10}\left(\frac{\sigma^2_{\mathrm{Y}_k}}{\sigma^2_{\mathrm{W}_k}}\right) = 10\log_{10}\left(\frac{E_Y}{E_W}\right) \tag{4}$$

2.2.2 Cramér-Rao Lower Bound: CRLB

An estimator is considered unbiased if its expected value is equal to the true value of the parameter it is estimating. The Cramér-Rao Lower Bound provides a theoretical lower bound on the variance of an unbiased estimator. In speech enhancement, it can be used to determine the theoretical limit of the performance of an estimator in terms of its variance.

$$\text{CRLB}(\theta) = \frac{1}{I(\theta)} \tag{5}$$

$$I(\theta) = -E\left[\frac{\partial^2 \ln f(X;\theta)}{\partial \theta^2}\right] \tag{6}$$

I(θ)is the Fisher Information for the parameter θ. $f(X;\theta)$ is the probability density function of the observed data X parameterized by θ. In this project, it's very hard to calculate the fisher information, so we use a general expression of CRLB. When it is a linear Gaussian model,like $x = h\theta + w, \quad w \sim \mathcal{N}(0, C_w)$, where θ is an unknown parameter, x is the observed signal, and w is Gaussian noise with a mean of 0 and a covariance matrix C_w. The CRLB can be simplified as $\frac{1}{\mathbf{h}^\mathrm{T}\mathbf{C}_w^{-1}\mathbf{h}}$. Additionally, since it is a zero-mean signal, we employ the Welch method to compute the average power, which in this case is equivalent to the variance.

2.2.3 Variance of the estimator

To determine the variance of the estimator, we use Equation 7 to calculate, where K and L denote the frequency-bin index and time frame, respectively. A lower variance signifies higher precision in the estimation process. Therefore, the closer the variance approaches the CRLB, the better the performance of the estimator.

$$\text{var}_{\text{emp}} = \frac{1}{KL}\sum_{k=1}^{K}\sum_{l=1}^{L}\left|\hat{S}(l,k) - S(l,k)\right|^2 \tag{7}$$

3 Deterministic speech signal

In this chapter, we assume that clean speech in noisy speech is the deterministic signal. The signal type of the clean speech determines how to estimate the clean method from the noisy speech.

3.1 Model selection

The common estimators for deterministic signals include Maximum Likelihood Estimation (MLE), Best Linear Unbiased Estimator (BLUE), and Least Squares (LS). Notably, MLE stands out for its robustness and versatility, requiring fewer distribution assumptions compared to BLUE and being less susceptible to outliers than LS. These features make MLE the preferred option for precise and reliable parameter estimation, particularly in diverse or intricate datasets like those found in speech signal data. In conclusion, we choose the MLE as the estimator to denoise the signal.

3.2 Implementation of MLE

The definition of MLE is to find the value $\mathbf{S}$ that maximizes $p(\mathbf{Y};\mathbf{S})$ for a fixed $\mathbf{Y}$. In the MLE method proposed in [1], the amplitude and phase spectra of the clean speech signal at frequency bin k, denoted as S_k and $\theta_s(k)$, are assumed to be uncertain but deterministic. Because the noise signal $W_m(l,k) \sim \mathcal{C}N(0, \sigma^2_{\mathrm{W}_m(l,k)})$, the distribution of $\mathbf{Y}(\omega_k)$ is also Gaussian distribution with variance $\sigma^2_{\mathrm{W}_m(l,k)}$ and mean $S_k e^{j}\theta_s(k)$. In the upcoming equations in this report, for the sake of simplicity in notation, we will focus on the estimation process in one time frame and one channel. Thus, the representation of the variance in noise can be denoted as $\sigma^2_{\mathrm{W}_k}$, signifying the variance of noise at frequency bin k. So, the probability density function of $\mathbf{Y}(\omega_k)$ can be formulated in Equation 8 [1].

$$p(Y(\omega_k); S_k, \theta_s(k)) = \frac{1}{\pi\sigma^2_{\mathrm{W}_k}}\exp\left[-\frac{\left|Y(\omega_k) - S_k e^{j}\theta_s(k)\right|^2}{\sigma^2_{\mathrm{W}_k}}\right] \tag{8}$$

To achieve the maximum estimation of $\mathbf{S}$, we need to simplify and differentiate the log-likelihood function $log(p(Y(\omega_k); S_k, \theta_s(k)))$ with respect to S_k and set the derivative to zero. Then, we can get the estimation function of S_k in Equation 9 [2]. In Equation 9, γ_k denotes the posteriori SNR which can be calculated by Equation 4. Because Equation 9 is solely designed to estimate the magnitude of $\hat{S}$, it is necessary to add the phase information of $\mathbf{Y}(\omega_k)$ after the magnitude estimation.

$$\hat{S}(\omega_k) = \left[\frac{1}{2} + \frac{1}{2}\sqrt{\frac{\gamma_k - 1}{\gamma_k}}\right] Y(\omega_k) \tag{9}$$

In equations Equation 9 and 4, the critical aspect lies in estimating the magnitude of $\hat{S}$ to compute the noise energy E_W. To address the variability in noise variance throughout the speech signal, voice activity detection (VAD) is employed to improve the estimation of the noise energy. VAD captures short-time energy from the input signal, comparing it to a threshold value [2], typically determined during non-speech periods. The step of VAD is outlined in algorithm 1. In algorithm 1, the parameter M is configured for the frames of the first second, considering the absence of speech during this initial period. The smoothing parameter μ is set to 0.9, falling within the recommended range of 0.3 to 0.96 [2]. Additionally, the SNR threshold of 5 dB is established to ensure accurate identification of speech frames [3].

To apply MLE across multiple channels, we choose to combine data from these channels by calculating the average value. Because the noise signal W_m has a zero-mean property, averaging across more channels makes the mean of Y approach zero. This can weaken the impact of noise on the speech signal to a certain extent, to achieve a certain degree of noise reduction purposes and convert the multi-channel data into one channel. Besides, the Python code to implement MLE is shown in Appendix B.

Algorithm 1: Voice Activity Detection

Data: Number of speech frames N, the first M frames

Set parameter value SNR threshold: θ, smoothing parameter: μ;

Calculate noise energy Calculate E_W as the average energy over the first M frames ($E_W = \sum_k \sigma^2_{W_k}$);

for $t = 1$ **to** N **do**

  Calculate energy of each frame E_Y;

  Calculate SNR $SNR = \frac{E_Y}{E_W}$;

  if $SNR < \theta$ **then**

    $E_W = \mu E_W + (1 - \mu) E_Y$

3.3 Results of MLE

In this section, we will initially present the results of MLE using only a single-channel data input. Then, we will showcase the MLE results in the context of a multi-channel case.

3.3.1 Case 1: MLE result in a single-channel

To test MLE in a single channel, we select the first channel from the noisy speech data. Following the MLE algorithm and VAD, we obtain the spectrum of enhanced speech in a dB scale, as illustrated in Figure 3. In **??**, it is clear that the speech signal is clearer compared to the original noisy speech (shown in Figure 2). Then, we reverse the FFT process to obtain the signal in the time domain to calculate the SNR based on Chapter 2.2.1. The enhanced speech SNR is found to be 3.325. This represents a 9.88% improvement compared to the original noisy data in the first channel (SNR=3.026).

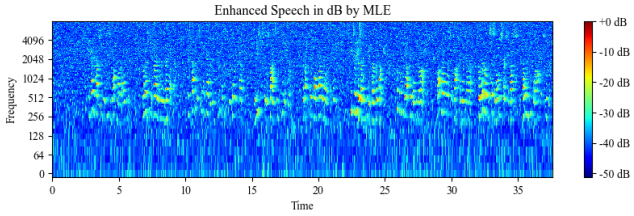

Figure 3: Enhanced speech spectrum of MLE in single-channel

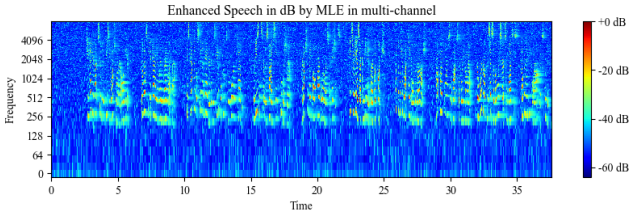

Figure 4: Enhanced speech spectrum of MLE in 16-channels

3.3.2 Case 2: MLE result in multi-channel

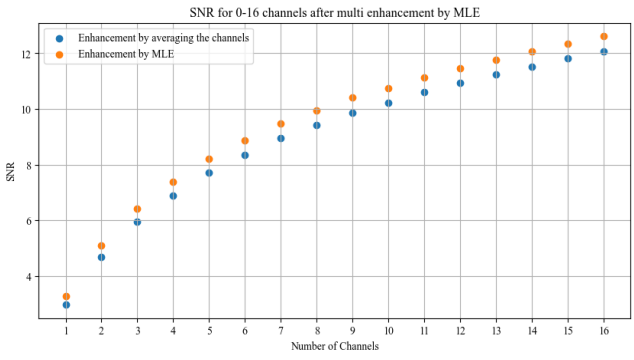

Figure 5: SNR for 1-16 channels after multi-channel enhancement by MLE

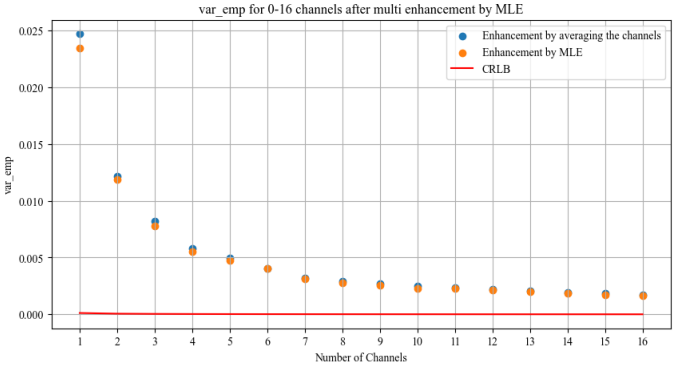

Figure 6: var_{emp} for 1-16 channels after multi-channel enhancement by MLE

After successfully applying MLE to denoise single-channel noisy speech, the next step involves testing MLE in a multi-channel environment. As outlined in Chapter 3.2, the multi-channel data is converted into a single-channel format by averaging. Following this conversion, the single-channel MLE process is employed to obtain the estimated clean speech. In Figure 4, the spectrum of the enhanced speech, utilizing data from all 16 channels, exhibits a clearer speech signal compared to both the original signal and the enhanced speech from a single channel. The effectiveness of MLE is evident in improving the quality of the speech signal. In terms of SNR, as shown in Figure 5, the orange dots representing MLE's SNR across 0 to 16 channels consistently outperform the blue dots, which represent SNR obtained solely by averaging multi-channel data. This highlights MLE's superiority in enhancing SNR

compared to the simple averaging method. Notably, the SNR of MLE increases with the growing number of channels, reaching its highest value of 12.61 when utilizing all 16 channels.

The variance of the estimator across 1 to 16 channels is depicted in Figure 6 based on the calculation of Equation 7. The blue dots illustrate the enhancement achieved by averaging the data, while the orange dots represent the MLE in multi-channel estimation. The red line signifies the CRLB, representing the minimum variance from the estimator. In Figure 6, it's clear that the variance of the MLE estimator decreases and tends to approach the CRLB as the number of microphones increases. Nonetheless, there remains a discernible deviation from the CRLB. Moreover, the variance of the MLE is slightly lower than that achieved by simply averaging the data. This observation suggests that the MLE has the potential to bring about a slight improvement in the variance compared to the simple averaging of the data.

4 Stochastic speech signal

In this section, we assume that the speech signal is not deterministic but rather a statistical signal. Therefore, we cannot continue to use MLE to reduce noise which is suitable to the deterministic signal. To estimate the random speech signal, we need an estimator that considers the statistical properties of both the signal and the noise.

4.1 Model selection

The common estimator of the random signal is the Maximum A Posteriori Probability (MAP) estimator and Minimum Mean Square Error (MMSE) estimator. The MAP estimator, emphasizes the use of prior information. However, MMSE often outperforms MAP in practice. This is because MMSE is more flexible in modeling noise and demonstrates better adaptability to the variations in actual noise, thereby offering more stable and accurate results in the process of speech signal restoration. As the clean speech is unknown in the speech noise reduction progress. It is hard to use the prior information to estimate the clean signal by MAP estimator. In conclusion, we decide to use the MMSE to denoise the noisy speech.

4.2 Implementation of MMSE

The goal of the MMSE algorithm is to minimize the expected value of the Bayesian mean square error (Bmse). After setting the derivative of Bmse to zero we obtain, we can get the estimated value $\hat{S}_k = E[S_k|Y_k]$. This equation is to calculate the mean of the a posteriori probability density function of S which is not easy because the speech signal is neither a stationary nor an ergodic process [2]. After computation, the gain function for estimating the magnitude of S_k in Equation 10 is derived [2]. This process is based on two assumptions: 1. Fourier transform coefficients exhibit a Gaussian probability distribution with a zero mean. 2. The Fourier transform coefficients are statistically independent.

$$\hat{S}_k = \frac{\sqrt{\pi}}{2}\frac{\sqrt{\nu_k}}{\gamma_k}\exp\left(-\frac{\nu_k}{2}\right)\left[(1+\nu_k)\,I_0\left(\frac{\nu_k}{2}\right)+\nu_k I_1\left(\frac{\nu_k}{2}\right)\right]Y_k \tag{10}$$

In Equation 10, I_0 and I_1 denote the modified Bessel functions of zero and first order, respectively. The variable ν_k is composed of the priori SNR ξ_k and the posteriori SNR γ_k, as defined in Equation 11. The equation for computing ξ_k is presented in Equation 12 ($\lambda_S(k)$ denotes the variance of clean speech signal), while the equation for calculating γ_k is given in Equation 4. In Equation 10, ξ_k plays a primary role in noise suppression, and the MMSE method is particularly sensitive to the fluctuations of ξ_k [2]. Therefore, an accurate and relatively smooth estimation of ξ_k is essential.

$$\nu_k = \frac{\xi_k}{1+\xi_k}\gamma_k \tag{11}$$

$$\xi_k = \frac{\lambda_S(k)}{\sigma^2_{\mathrm{W}_k}} \tag{12}$$

To estimate ξ_k, we can use its relationship with γ_k called decision-directed method [4]. The equation to calculate ξ_k by the decision-directed method is shown in Equation 13. In this equation, the priori SNR ξ_k in time frame l is updated using the value from the previous time frame $(l-1)$ and the posteriori SNR γ_k in current time frame l. The parameter α is called the smoothing constant and is typically set to 0.98, as recommended in [2]. The value $\xi_{\min}$ is important for reducing low-level musical noise and is advised to be set to $-15dB$ according to [5]. When determining the posteriori SNR γ_k, the VAD method can be employed for estimation, utilizing Equation 4 and following algorithm 1.

$$\hat{\xi}_k(l) = \max\left[\alpha\frac{\hat{S}_k^2(l-1)}{\sigma^2_{\mathrm{W}_k}(l-1)} + (1-\alpha)\max[\gamma_k(l)-1,0], \xi_{\min}\right] \tag{13}$$

Similar to the approach employed in the MLE, we can calculate the average value across all channels. This process helps transform the multi-channel signal into a single-channel signal, and the detailed procedure is outlined in Chapter 3.2. Besides, the Python code to implement MLE is shown in Appendix B.

4.3 Results of MMSE

4.3.1 Case 1: MMSE result in a single-channel

Like the test for the first channel in Chapter 3.3.1, after we estimate the first channel noisy speech signal based on MMSE, VAD and the decision-directed method, we can get the spectrum of the enhanced speech. The spectrum is shown in Figure 7. We can find that the speech signal can be recovered from the noisy data successfully. After reversing the FFT process, we calculated the

SNR following the steps outlined in Chapter 3.3.1. The SNR for MMSE in the single-channel is 9.57, which is almost three times higher than the SNR for MLE in the single-channel test (SNR = 3.325). Comparing the results of MLE and MMSE, we find that MMSE performs better in reducing noise in the same test environment.

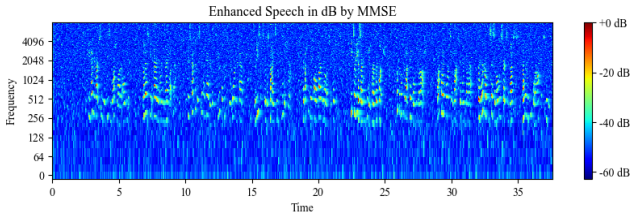

Figure 7: Enhanced speech spectrum of MMSE in single-channel

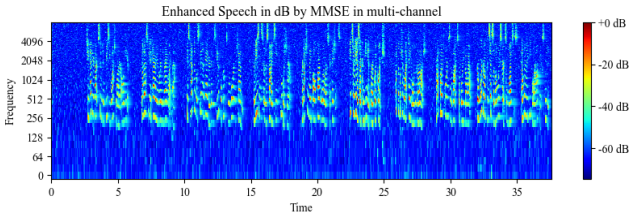

Figure 8: Enhanced speech spectrum of MMSE in 16-channels

4.3.2 Case 2: MMSE result in multi-channel

In the multi-channel test, similar to the single-channel MMSE test and the MLE multi-channel test, we averaged signals across varying channel numbers (1 to 16) and applied the MMSE algorithm for clean speech estimation. The resulting spectrum in Figure 8 demonstrates that the speech signal is noticeably clearer compared to the 16-channel MLE spectrum (Figure 4). In terms of SNR, as shown in Figure 9, the 16-channels MMSE significantly outperforms both 16-channels MLE and the average of 16 channels. Notably, MMSE achieves this high performance with just two microphones, rivaling the effectiveness of MLE with all 16 channels. Moreover, SNR increases with additional channels, peaking at 16 channels with a value of 23.008, nearly double that of MLE with 16 channels. The variance analysis in Figure 10 reveals that MMSE consistently maintains significantly lower variance compared to MLE and the average data. With the increasing number of channels, MMSE's variance decreases and nearly approaches the CRLB. Besides, the variance in one-channel MMSE even surpasses that of 16-channels MLE. The graph in Appendix A further illustrates that the difference between 16-channels MMSE and CRLB is numerically less than 0.0002.

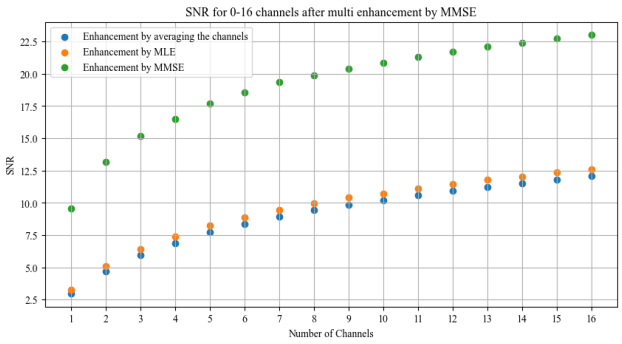

Figure 9: SNR for 1-16 channels after multi-channel enhancement by MMSE

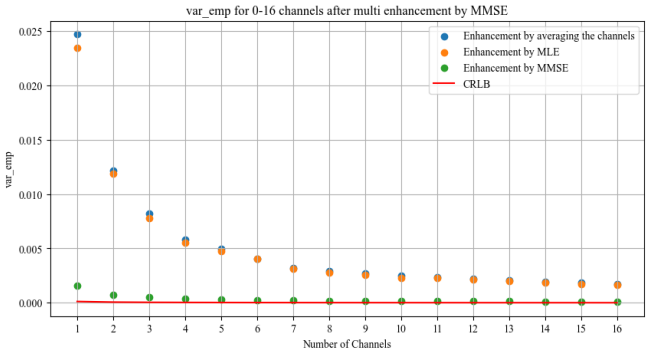

Figure 10: var_{emp} for 1-16 channels after multi-channel enhancement by MMSE

5 Conclusion

In this report, we first introduce the background and theoretically analyze the speech system. Next in the following two scenarios—Deterministic Speech Signal and Stochastic Speech Signal—we conduct a comparative analysis of various models, elucidating our rationale for selecting the MLE and the MMSE methods. When the speech signal is deterministic, we choose MLE as our estimator because MLE is more robust and flexible than BLUE and LS, needing fewer assumptions and being less affected by outliers. When the speech signal is Stochastic, we choose MMSE because of its flexibility in noise modeling and better adaptability to noise variations. What's more, the MAP estimator's reliance on prior information is less effective in the context of unknown clean speech during noise reduction.

In the solution of this problem, We use the variance of the estimator to assess our model and test if it reaches the Cramer-Rao bound. What's more, in order to measure the clarity of a signal in the presence of noise, We use SNR to assess our model. After preprocessing the data, we use VAD to extract the power of the noise and apply our estimator to the speech signal. Then We detail the implementation process of speech enhancement.

As a result, we can conclude that simply combining the data from one to sixteen channels has an improvement in SNR and variance. In the scenario of a deterministic signal, our MLE estimator performs better than just combining the data but doesn't reach the CRLB. However, when we use MMSE in the scenario of stochastic signal, the estimator performs much better than MLE, and it is very close to the CRLB. We assume the reason is that speech signals are generally considered as random signals in reality. MMSE's superiority in speech enhancement applications primarily stems from its suitability for handling stochastic signals like speech, as well as its ability to adapt to variations in speech signals in noisy environments. MLE, on the other hand, might be more effective in other applications where the signals are more deterministic and noise is less of a factor.

References

[1] McAulay, R. J. and Malpass, M. L. (1980), Speech enhancement using a soft-decision noise suppression filter, IEEE Trans. Acoust. Speech Signal Process., 28, 137–145

[2] Philipos C. Loizou. 2013. Speech Enhancement: Theory and Practice (2nd. ed.). CRC Press, Inc., USA.

[3] T. Bäckström, "Voice activity detection (VAD)," 8.1. Voice Activity Detection (VAD) - Introduction to Speech Processing, https://speechprocessingbook.aalto.fi/Recognition/Voice_activity_detection.html (accessed Jan. 4, 2024).

[4] Ephraim, Y. and Malah, D. (1984), Speech enhancement using a minimum mean square error short-time spectral amplitude estimator, IEEE Trans. Acoust. Speech Signal Process., 32(6), 1109–1121

[5] Cappe, O. (1994), Elimination of the musical noise phenomenon with the Ephraim and Malah noise suppressor, IEEE Trans. Speech Audio Process., 2(2), 346–349.

Appendix A Zoomed-in graph of the variance of 16-channel MMSE and CRLB

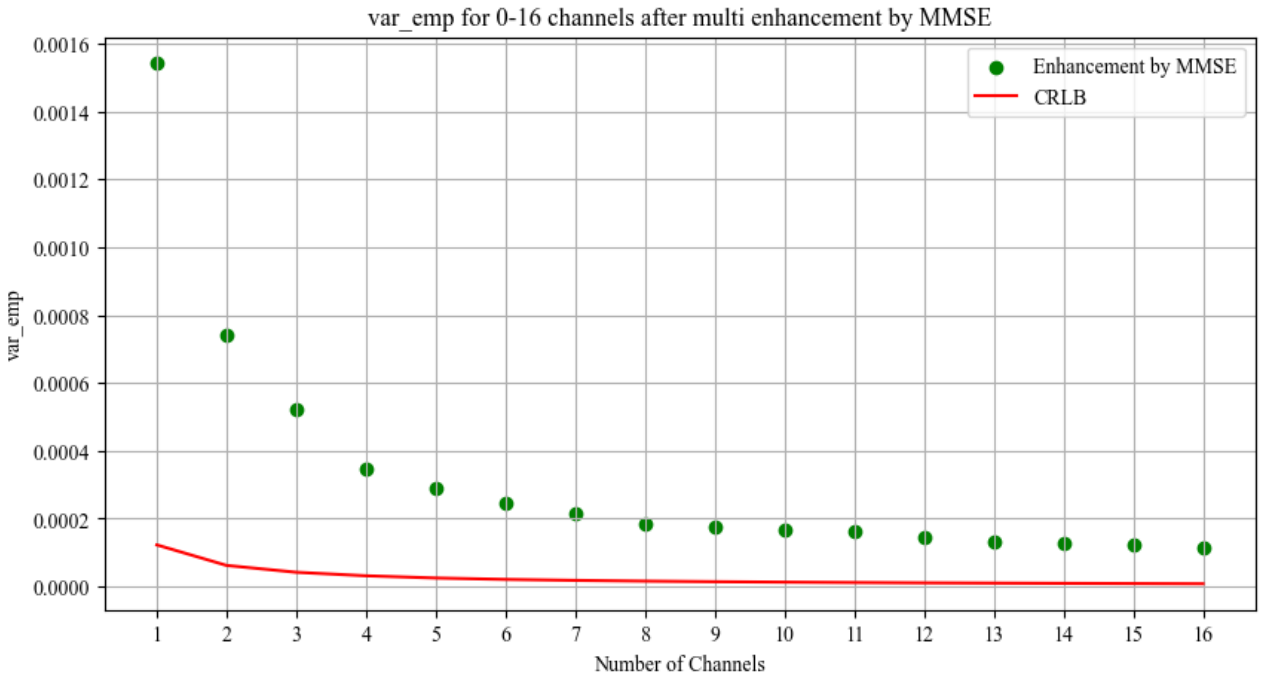

Figure 11: Zoomed-in graph of variance of 16-channel MMSE and CRLB

Appendix B Program code in python

```
##################################################

# # Multi-Microphone Speech Enhancement

##################################################
# ***ET4386 Estimation and Detection Project***
#
# *Author: Yanqi Hong | Kaishen Lin*
##################################################

#import library
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import numpy as np
from scipy.signal import hamming
from scipy.signal import welch
from scipy.fft import fft
from scipy.fftpack import fft
import librosa
import scipy.special as sp
import soundfile as sf
import statistics

# Set the default font for plots
plt.figure(dpi=200)
plt.rcParams['font.family'] = 'Times New Roman'



##################################################
# ## 1. Data analysis
##################################################

#-----------------------------------------#
# ### 1.1 Data input and general analysis
#-----------------------------------------#

#########################################################
####Code to conver .mat file to .csv file in matlab#####
# writematrix(Clean,'Clean.csv');
# writematrix(Data,'Data.csv');
#########################################################

#import csv file
Data = pd.read_csv('Data.csv', header=None)      #noisy speech with 16 channels
Clean=pd.read_csv('Clean.csv', header=None)      #clean speech
display(Data)
display(Clean)
# plot all 16 channels in 16 subplots
plt.figure(figsize=(20,10))
for i in range(1,17):
    t=np.linspace(0, len(Data[i-1])/fs, len(Data[i-1]))
    plt.subplot(4,4,i)
    plt.plot(Data[i-1], linewidth=0.5)  # Set the linewidth to 0.5
    plt.title('Channel '+str(i))
    plt.xlabel('Time (s))')
    plt.ylabel('Amplitude')

plt.subplots_adjust(hspace=0.5, wspace=0.5)
plt.show()

# plot clean speech
t=np.linspace(0, len(Clean)/fs, len(Clean))
plt.figure(figsize=(10,5))
plt.plot(t,Clean, linewidth=0.5)  # Set the linewidth to 0.5
plt.title('Clean Speech')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.show()

Clean = np.transpose(Clean.to_numpy())[0]
Data = Data.to_numpy()


#-----------------------------------------#
# ### 1.2 SNR evaluation
```

```
#-----------------------------------------#

# Evaluation by SNR
def SNR(NoisyData):
    """
    Calculate the Signal-to-Noise Ratio (SNR) of the given noisy data.

    Parameters:
    NoisyData (array-like): The input noisy data.

    Returns:
    float: The calculated SNR value.
    """
    noise_segment = NoisyData[:15000]
    snr = 10*np.log10(np.var(NoisyData)/np.var((noise_segment)))
    return snr




# Calculate SNR for each channel
SNR_each_channel = np.zeros(16)
for i in range(16):
    SNR_each_channel[i] = SNR(Data[:,i])

# Plot SNR for each channel as points
plt.figure(figsize=(10,5))
plt.scatter(range(1, 17), SNR_each_channel)
plt.title('original SNR for each channel')
plt.xticks(range(1, 17))
plt.xlabel('Channel')
plt.ylabel('SNR')
plt.grid()
plt.show()

print('average SNR : ', statistics.mean(SNR_each_channel))


#-----------------------------------------#
# ### 1.3 Spectrogram analysis
#-----------------------------------------#

#specrum in clean speech
Fs = 16000
# speech_noisy_freq = np.zeros((320, 3760, 1))


sample_rate = 16000
frame_size = 2*int(sample_rate * 0.02)
hop_size = frame_size // 4

clean_freq = librosa.stft(Clean, n_fft=frame_size, hop_length=hop_size, window='hann')


# subplots of the clean speech in linear and dB scale spectrogram
plt.figure(figsize=(10,5))
plt.subplot(2,1,1)
librosa.display.specshow(abs(clean_freq), sr=sample_rate, hop_length=hop_size, x_axis='time', y_axis='log',cmap='jet')
plt.title('Clean Speech in linear scale')
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.colorbar()
plt.subplots_adjust(hspace=0.5, wspace=0.5)


plt.subplot(2,1,2)
librosa.display.specshow(librosa.amplitude_to_db(abs(clean_freq), ref=np.max), sr=sample_rate, hop_length=hop_size, x_axis=
'time', y_axis='log',cmap='jet')
plt.title('Clean Speech in dB')
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.colorbar(format='%+2.0f dB')
plt.show()


#specrum in noisy speech
Fs = 16000
speech_noisy_freq = np.zeros((321, 3761, 16))

```

```
#calcuate the fft for each channel
for i in range(16):
    speech_noisy_freq[:,:,i] = librosa.stft(Data[:,i], n_fft=frame_size, hop_length=hop_size, window='hann')


# # subplots of the noisy speech (channel 1) in linear and dB scale spectrogram
plt.figure(figsize=(10,5))
plt.subplot(2,1,2)
librosa.display.specshow(librosa.amplitude_to_db(abs(speech_noisy_freq[:,:,0]), ref=np.max), sr=sample_rate, hop_length=hop_size, x_axis='time', y_axis='log',cmap='jet')
plt.title('Original noisy Speech (channel 1) in dB')
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.colorbar(format='%+2.0f dB')

plt.subplot(2,1,1)
librosa.display.specshow(abs(speech_noisy_freq[:,:,0]), sr=sample_rate, hop_length=hop_size, x_axis='time', y_axis='log', cmap='jet')
plt.title('Original noisy Speech (channel 1) in linear scale')
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.colorbar()
plt.subplots_adjust(hspace=0.5, wspace=0.5)
plt.show()


# subplots for all 16 channels in dB scale spectrogram
plt.figure(figsize=(45,25))
for i in range(16):
    plt.subplot(4,4,i+1)
    librosa.display.specshow(librosa.amplitude_to_db(abs(speech_noisy_freq[:,:,i]), ref=np.max), sr=sample_rate, hop_length=hop_size, x_axis='time', y_axis='log',cmap='jet')
    plt.title('Channel '+str(i+1))
    plt.xlabel('Time')
    plt.ylabel('Frequency')
    plt.colorbar(format='%+2.0f dB')

plt.subplots_adjust(hspace=0.5, wspace=0.25)  # Adjust hspace to reduce the amount of whitespace below the main title
plt.suptitle('Noisy Speech in dB for all 16 channels')
plt.show()

plt.figure(figsize=(45,25))
for i in range(16):
    plt.subplot(4,4,i+1)
    librosa.display.specshow(abs(abs(speech_noisy_freq[:,:,i])), sr=sample_rate, hop_length=hop_size, x_axis='time', y_axis='log',cmap='jet')
    plt.title('Channel '+str(i+1))
    plt.xlabel('Time')
    plt.ylabel('Frequency')
    plt.colorbar()


plt.subplots_adjust(hspace=0.5, wspace=0.25)    # Adjust hspace to reduce the amount of whitespace below the main title
plt.suptitle('Noisy Speech in linear scale for all 16 channels')
plt.show()



#-----------------------------------------#
# ### 1.4 CRLB analysis
#-----------------------------------------#

#Enancement for all channels only by calculating the mean of all channels
snr_all_mean=np.zeros(16)
speech_enhanced_all_mean=np.zeros((601600,16))
speech_enhanced_freq_all_mean=np.zeros((321,3761,16))

for i in range(16):
    speech_enhanced_freq_all_mean[:,:,i]=np.mean(speech_noisy_freq[:,:,0:i+1],axis=2)
    speech_enhanced_all_mean[:,i] = librosa.istft(speech_enhanced_freq_all_mean[:,:,i], n_fft=frame_size, hop_length=hop_size, window='hann')
    snr_all_mean[i] = SNR(speech_enhanced_all_mean[:,i])


# #CRLB
CRLB=np.zeros(16)
fs=16000
for i in range(16):
    channel_speech=np.mean(Data[:,0:i+1],axis=1)
    noise_segment = channel_speech[:fs]
    frequencies, power_spectral_density = welch(noise_segment, fs, nperseg=320) #Calculate power spectral density
```

```
    CRLB[i] = np.sum(power_spectral_density) / len(power_spectral_density)  #Calculate average power


# Plot CRLB for each channel as points
plt.figure(figsize=(10,5))
plt.scatter(range(1, 17), CRLB)
plt.title('CRLB for channel 0-16')
plt.xticks(range(1, 17))
plt.xlabel('Channel')
plt.ylabel('CRLB')
plt.grid()
plt.show()



################################################
# ## 2. Denoise
################################################

#-----------------------------------------#
# ### 2.1. Denoise with MLE
#-----------------------------------------#


#.......................................#
# #### 2.1.1 MLE for single channel
#.......................................#

# #### Estimation
def MLE_speech_enhancement(speech_noisy_freq):
    """
    Perform maximum likelihood estimation (MLE) based speech enhancement.

    Parameters:
    speech_noisy_freq (ndarray): Noisy speech frequency domain representation with shape (D, T), where D is the number of frequency bins, T is the number of frames.

    Returns:
    speech_enhanced (ndarray): Enhanced speech in the time domain.
    speech_enhanced_freq (ndarray): Enhanced speech frequency domain representation with shape (D, T), where D is the number of frequency bins and T is the number of frames.

    """

    # parameters
    max_gamma = 40  # maximum value of gamma
    mu_VAD = 0.9  # parameter for VAD noise tracking
    th_VAD =15  # VAD decision threshold


    phase_noisy = np.angle(speech_noisy_freq)
    mag_noisy = np.abs(speech_noisy_freq)

    D, T = np.shape(mag_noisy)


    time_index_num = int(1//0.02)    # initial noise length: 1s
    noise_segment = mag_noisy[:, 0:time_index_num]
    mag_noise = np.mean(np.abs(noise_segment), axis=1)
    power_noise = mag_noise**2

    mag_enhance = np.zeros([D, T])

    for i in range(T):

        # get energy spectrum and magnitude spectrum for each frame
        mag_frame = mag_noisy[:, i]
        power_frame = mag_frame**2

        # get Signal-to-Noise Ratio (SNR) used for VAD calculation
        SNR_VAD = 10 * np.log10(np.sum(power_frame) / np.sum(power_noise))

        # calculate posterior SNR
        gamma = np.minimum(power_frame / power_noise, max_gamma)

        # update power_noise based on VAD
        mu = mu_VAD
        if SNR_VAD < th_VAD:
            power_noise = mu * power_noise + (1 - mu) * power_frame


```

```
        G = 0.5 + 0.5 * np.sqrt((abs(gamma - 1)) / gamma)


        mag_enhance_frame = G * mag_frame
        mag_enhance[:, i] = mag_enhance_frame

    speech_enhanced_freq = mag_enhance * np.exp(1j * phase_noisy)

    speech_enhanced = librosa.istft(speech_enhanced_freq, n_fft=frame_size, hop_length=hop_size, window='hann')

    return speech_enhanced, speech_enhanced_freq


# Denoise the first noisy speech channel by the MLE speech enhancement
enhanced_speech, enhanced_speech_freq = MLE_speech_enhancement(speech_noisy_freq[:,:,0])


# #### Evaluation
print("SNR of the enhanced speech is: ", SNR(enhanced_speech))

#plot the enhanced speech
#subplots
plt.figure(figsize=(10,5))
plt.subplot(2,1,1)
librosa.display.specshow(librosa.amplitude_to_db(abs(speech_noisy_freq[:,:,0]), ref=np.max), sr=sample_rate, hop_length=hop_size, x_axis='time', y_axis='log',cmap='jet')
plt.title('Original noisy Speech in dB')
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.colorbar(format='%+2.0f dB')

plt.subplot(2,1,2)
librosa.display.specshow(librosa.amplitude_to_db(abs(enhanced_speech_freq), ref=np.max), sr=sample_rate, hop_length=hop_size, x_axis='time', y_axis='log',cmap='jet')
plt.title('Enhanced Speech in dB by MLE')
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.colorbar(format='%+2.0f dB')
plt.subplots_adjust(hspace=0.5, wspace=0.5)
plt.show()

sf.write("enhce_mmse.wav", enhanced_speech, fs)




#.....................................#
# #### 2.1.2. MLE for multi channel
#.....................................#

# #### Estimation
def MLE_multi_channel_speech_enhancement(speech_noisy_freq):
    """
    Applies Maximum Likelihood Estimation (MLE) based multi-channel speech enhancement to the given noisy speech frequency domain representation.

    Parameters:
    speech_noisy_freq (ndarray): Noisy speech frequency domain representation with shape (D, T, L), where D is the number of frequency bins, T is the number of frames, and L is the number of channels.

    Returns:
    speech_enhanced (ndarray): Enhanced speech in the time domain.
    speech_enhanced_freq (ndarray): Enhanced speech frequency domain representation with shape (D, T), where D is the number of frequency bins and T is the number of frames.
    """


    speech_noisy_freq = np.mean(speech_noisy_freq, axis=2)  # average over all channels

    # parameters
    max_gamma = 40  # maximum value of gamma
    mu_VAD = 0.9  # parameter for VAD noise tracking
    th_VAD = 5  # VAD decision threshold


    phase_noisy = np.angle(speech_noisy_freq)
    mag_noisy = np.abs(speech_noisy_freq)

    D, T = np.shape(mag_noisy)

    time_index_num = int(1//0.02)    # initial noise length: 1s
```

```
    noise_segment = mag_noisy[:, 0:time_index_num]
    mag_noise = np.mean(np.abs(noise_segment), axis=1)
    power_noise = mag_noise**2

    mag_enhance = np.zeros([D, T])

    for i in range(T):

        # get energy spectrum and magnitude spectrum for each frame
        mag_frame = mag_noisy[:, i]
        power_frame = mag_frame**2

        # get Signal-to-Noise Ratio (SNR) used for VAD calculation
        SNR_VAD = 10 * np.log10(np.sum(power_frame) / np.sum(power_noise))

        # calculate posterior SNR
        gamma = np.minimum(power_frame / power_noise, max_gamma)

        # update power_noise based on VAD
        mu = mu_VAD
        if SNR_VAD < th_VAD:
            power_noise = mu * power_noise + (1 - mu) * power_frame


        G = 0.5 + 0.5 * np.sqrt((abs(gamma - 1)) / gamma)
        mag_enhance_frame = G * mag_frame
        mag_enhance[:, i] = mag_enhance_frame

    speech_enhanced_freq = mag_enhance * np.exp(1j * phase_noisy)

    speech_enhanced = librosa.istft(speech_enhanced_freq, n_fft=frame_size, hop_length=hop_size, window='hann')

    return speech_enhanced, speech_enhanced_freq


# Denoise the noisy speech by 16-channel MLE speech enhancement
enhanced_speech, enhanced_speech_freq = MLE_multi_channel_speech_enhancement(speech_noisy_freq)



# #### Evaluation

# var_emp evaluation
def Var_emp(enhanced_speech_freq_all):
    """
    Calculate the empirical variance for each frame in the enhanced speech frequency.

    Parameters:
    enhanced_speech_freq_all (ndarray): Array of shape (D, T, L) representing the enhanced speech frequency.

    Returns:
    var_emp_all (ndarray): Array of shape (L,) containing the empirical variance for each frame.
    """
    var_emp_all=np.zeros(16)
    D,T,L = np.shape(enhanced_speech_freq_all)
    for i in range(L):

        for j in range(T):
            var=np.sum((np.abs(clean_freq[:,j]-enhanced_speech_freq_all[:,j,i]))**2)

        var_emp_all[i] = (1/(D*T))*np.sum(var)
    return var_emp_all


#Enancement for all channels only by calculating the mean of all channels
snr_all_mean=np.zeros(16)
speech_enhanced_all_mean=np.zeros((601600,16))
speech_enhanced_freq_all_mean=np.zeros((321,3761,16))

for i in range(16):
    speech_enhanced_freq_all_mean[:,:,i]=np.mean(speech_noisy_freq[:,:,0:i+1],axis=2)
    speech_enhanced_all_mean[:,i] = librosa.istft(speech_enhanced_freq_all_mean[:,:,i], n_fft=frame_size, hop_length=
hop_size, window='hann')
    snr_all_mean[i] = SNR(speech_enhanced_all_mean[:,i])


print("SNR of the enhanced speech is: ", SNR(enhanced_speech))
#plot the enhanced speech
#subplots
plt.figure(figsize=(10,5))
```

```
plt.subplot(2,1,1)
librosa.display.specshow(librosa.amplitude_to_db(abs(speech_noisy_freq[:,:,0]), ref=np.max), sr=sample_rate, hop_length=hop_size, x_axis='time', y_axis='log',cmap='jet')
plt.title('Original noisy Speech in dB')
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.colorbar(format='%+2.0f dB')

plt.subplot(2,1,2)
librosa.display.specshow(librosa.amplitude_to_db(abs(enhanced_speech_freq), ref=np.max), sr=sample_rate, hop_length=hop_size, x_axis='time', y_axis='log',cmap='jet')
plt.title('Enhanced Speech in dB by MLE in multi-channel')
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.colorbar(format='%+2.0f dB')
plt.subplots_adjust(hspace=0.5, wspace=0.5)
plt.show()

sf.write("enhce_mmse.wav", enhanced_speech, fs)


#Evaluion for 0-16 channels by SNR and var_emp
enhanced_speech_all_mle=np.zeros((len(enhanced_speech),16))
enhanced_speech_freq_all_mle=np.zeros((321,3761,16))
snr_all_mle=np.zeros(16)

for i in range(16):
    enhanced_speech_all_mle[:,i], enhanced_speech_freq_all_mle[:,:,i] = MLE_multi_channel_speech_enhancement(speech_noisy_freq[:,:,0:i+1])
    snr_all_mle[i] = SNR(enhanced_speech_all_mle[:,i])


#plot the enhanced speech for all 16 channels
#subplots
plt.figure(figsize=(45,25))
for i in range(16):
    plt.subplot(4,4,i+1)
    librosa.display.specshow(librosa.amplitude_to_db(abs(enhanced_speech_freq_all_mle[:,:,i]), ref=np.max), sr=sample_rate, hop_length=hop_size, x_axis='time', y_axis='log',cmap='jet')
    plt.title(str(i+1)+' Channels')
    plt.xlabel('Time')
    plt.ylabel('Frequency')
    plt.colorbar(format='%+2.0f dB')


#plot snr for each channel
print("SNR of the enhanced speech for all 0-16 channels is: ", snr_all_mle)
plt.figure(figsize=(10,5))
plt.scatter(range(1, 17), snr_all_mean)
plt.scatter(range(1, 17), snr_all_mle)
plt.title('SNR for 0-16 channels after multi enhancement by MLE')
plt.xticks(range(1, 17))
plt.xlabel('Number of Channels')
plt.ylabel('SNR')
plt.grid()
#legand
plt.legend(['Enhancement by averaging the channels','Enhancement by MLE'])
plt.show()


# calculate the Var_emp
var_emp_all_mle=Var_emp(enhanced_speech_freq_all_mle)
var_emp_all_mean=Var_emp(speech_enhanced_freq_all_mean)
print("var_emp of the enhanced speech for all 0-16 channels is: ", var_emp_all_mle)
plt.figure(figsize=(10,5))
plt.scatter(range(1, 17), var_emp_all_mean)
plt.scatter(range(1, 17), var_emp_all_mle)
plt.plot(range(1, 17), CRLB, color='red')
plt.title('var_emp for 0-16 channels after multi enhancement by MLE')
plt.xticks(range(1, 17))
plt.xlabel('Number of Channels')
plt.ylabel('var_emp')
plt.legend(['Enhancement by averaging the channels','Enhancement by MLE','CRLB'])
plt.grid()
plt.show()
```

```
544
545
546
547 #-----------------------------------------#
548 # ### 2.2 Denoise with MMSE
549 #-----------------------------------------#
550
551
552 #.......................................#
553 # #### 2.2.1 MMSE for single channel
554 #.......................................#
555
556 # #### Estimation
557 #single channel speech enhancement
558 def MMSE_speech_enhancement(speech_noisy_freq):
559     """
560     Enhances the given noisy speech signal using the MMSE (Minimum Mean Square Error) algorithm.
561
562     Parameters:
563     speech_noisy_freq (ndarray): Noisy speech frequency domain representation with shape (D, T), where D is the number of
    frequency bins, T is the number of frames.
564
565     Returns:
566     speech_enhanced (ndarray): Enhanced speech in the time domain.
567     speech_enhanced_freq (ndarray): Enhanced speech frequency domain representation with shape (D, T), where D is the
    number of frequency bins and T is the number of frames.
568     """
569
570
571
572     # parameters
573     max_gamma = 40  # maximum value of gamma
574     alpha = 0.98  # parameter for updating ksi using decision-direct
575
576     ksi_min = 10 ** (-15 / 10)  # minimum value of ksi: -15dB
577                                 #recommended in literature as -15dB
578
579     mu_VAD = 0.9  # parameter for VAD noise tracking
580     th_VAD = 5  # VAD decision threshold
581
582     phase_noisy = np.angle(speech_noisy_freq)
583     mag_noisy = np.abs(speech_noisy_freq)
584
585     D, T = np.shape(mag_noisy)
586
587     time_index_num = int(1//0.02)    # initial noise length: 1s
588     noise_segment = mag_noisy[:, 0:time_index_num]
589     mag_noise = np.mean(np.abs(noise_segment), axis=1)
590     power_noise = mag_noise**2
591
592     mag_enhance = np.zeros([D, T])
593     alpha = alpha
594
595     for i in range(T):
596
597         # get energy spectrum and magnitude spectrum for each frame
598         mag_frame = mag_noisy[:, i]
599         power_frame = mag_frame**2
600
601         # get Signal-to-Noise Ratio (SNR) used for VAD calculation
602         SNR_VAD = 10 * np.log10(np.sum(power_frame) / np.sum(power_noise))
603
604         # calculate posterior SNR
605         gamma = np.minimum(power_frame / power_noise, max_gamma)
606
607         # calculate prior SNR
608         if i == 0:
609             ksi = alpha + (1 - alpha) * np.maximum(gamma - 1, 0)
610         else:
611             ksi = alpha * power_enhance_frame / power_noise + (1 - alpha) * np.maximum(gamma - 1, 0)
612             # limit the minimum value of ksi
613             ksi = np.maximum(ksi_min, ksi)
614
615         # update power_noise based on VAD
616         mu = mu_VAD
617         if SNR_VAD < th_VAD:
618             power_noise = mu * power_noise + (1 - mu) * power_frame
619         c = np.sqrt(np.pi) / 2
620
621         v = gamma * ksi / (1 + ksi)
622
```

```
        # standard mmse
        j_0 = sp.iv(0, v/2)
        j_1 = sp.iv(1, v/2)
        C = np.exp(-0.5 * v)
        H = (((c * (v ** 0.5)) * C) / gamma)  * ((1 + v) * j_0 + v * j_1 )

        mag_enhance_frame = H * mag_frame
        mag_enhance[:, i] = mag_enhance_frame

        power_enhance_frame = mag_enhance_frame ** 2

    speech_enhanced_freq = mag_enhance * np.exp(1j * phase_noisy)

    speech_enhanced = librosa.istft(speech_enhanced_freq, n_fft=frame_size, hop_length=hop_size, window='hann')

    return speech_enhanced, speech_enhanced_freq



# Try to use the first channel to do the speech enhancement
enhanced_speech, enhanced_speech_freq = MMSE_speech_enhancement(speech_noisy_freq[:,:,0])


# #### Evaluation
print("SNR of the enhanced speech is: ", SNR(enhanced_speech))

#plot the enhanced speech
#subplots
plt.figure(figsize=(10,5))
plt.subplot(2,1,1)
librosa.display.specshow(librosa.amplitude_to_db(abs(speech_noisy_freq[:,:,0]), ref=np.max), sr=sample_rate, hop_length=hop_size, x_axis='time', y_axis='log',cmap='jet')
plt.title('Original noisy Speech in dB')
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.colorbar(format='%+2.0f dB')

plt.subplot(2,1,2)
librosa.display.specshow(librosa.amplitude_to_db(abs(enhanced_speech_freq), ref=np.max), sr=sample_rate, hop_length=hop_size, x_axis='time', y_axis='log',cmap='jet')
plt.title('Enhanced Speech in dB by MMSE')
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.colorbar(format='%+2.0f dB')
plt.subplots_adjust(hspace=0.5, wspace=0.5)
plt.show()

sf.write("enhce_mmse.wav", enhanced_speech, fs)


#.....................................#
# #### 2.2.2 Multi channel MMSE
#.....................................#


# #### Estimation
#multi channel speech enhancement
def MMSE_MultiChannel_speech_enhancement(speech_noisy_freq):
    """
    Enhances the given noisy speech signal using the MMSE (Minimum Mean Square Error) algorithm.

    Parameters:
    speech_noisy_freq (ndarray): Noisy speech frequency domain representation with shape (D, T), where D is the number of frequency bins, T is the number of frames.

    Returns:
    speech_enhanced (ndarray): Enhanced speech in the time domain.
    speech_enhanced_freq (ndarray): Enhanced speech frequency domain representation with shape (D, T), where D is the number of frequency bins and T is the number of frames.
    """

    speech_noisy_freq=np.mean(speech_noisy_freq,axis=2) #average over all channels

    # parameters
    max_gamma = 40  # maximum value of gamma
    alpha = 0.98  # parameter for updating ksi using decision-direct

    ksi_min = 10 ** (-15 / 10)  # minimum value of ksi: -15dB
                                #recommended in literature as -15dB

```

```
    mu_VAD = 0.9  # parameter for VAD noise tracking
    th_VAD = 5  # VAD decision threshold

    phase_noisy = np.angle(speech_noisy_freq)
    mag_noisy = np.abs(speech_noisy_freq)

    D, T = np.shape(mag_noisy)

    time_index_num = int(1//0.02)    # initial noise length: 1s
    noise_segment = mag_noisy[:, 0:time_index_num]
    mag_noise = np.mean(np.abs(noise_segment), axis=1)
    power_noise = mag_noise**2

    mag_enhance = np.zeros([D, T])
    alpha = alpha

    for i in range(T):

        # get energy spectrum and magnitude spectrum for each frame
        mag_frame = mag_noisy[:, i]
        power_frame = mag_frame**2

        # get Signal-to-Noise Ratio (SNR) used for VAD calculation
        SNR_VAD = 10 * np.log10(np.sum(power_frame) / np.sum(power_noise))

        # calculate posterior SNR
        gamma = np.minimum(power_frame / power_noise, max_gamma)

        # calculate prior SNR
        if i == 0:
            ksi = alpha + (1 - alpha) * np.maximum(gamma - 1, 0)
        else:
            ksi = alpha * power_enhance_frame / power_noise + (1 - alpha) * np.maximum(gamma - 1, 0)
            # limit the minimum value of ksi
            ksi = np.maximum(ksi_min, ksi)

        # update power_noise based on VAD
        mu = mu_VAD
        if SNR_VAD < th_VAD:
            power_noise = mu * power_noise + (1 - mu) * power_frame

        c = np.sqrt(np.pi) / 2

        v = gamma * ksi / (1 + ksi)

        # standard mmse
        j_0 = sp.iv(0, v/2)
        j_1 = sp.iv(1, v/2)
        C = np.exp(-0.5 * v)
        H = (((c * (v ** 0.5)) * C) / gamma)  * ((1 + v) * j_0 + v * j_1 )

        mag_enhance_frame = H * mag_frame
        mag_enhance[:, i] = mag_enhance_frame

        power_enhance_frame = mag_enhance_frame ** 2

    speech_enhanced_freq = mag_enhance * np.exp(1j * phase_noisy)

    speech_enhanced = librosa.istft(speech_enhanced_freq, n_fft=frame_size, hop_length=hop_size, window='hann')

    return speech_enhanced, speech_enhanced_freq



enhanced_speech, enhanced_speech_freq = MMSE_MultiChannel_speech_enhancement(speech_noisy_freq)



# #### Evaluation
print("SNR of the enhanced speech is: ", SNR(enhanced_speech))


#plot the enhanced speech
plt.figure(figsize=(10,5))
plt.subplot(2,1,1)
librosa.display.specshow(librosa.amplitude_to_db(abs(speech_noisy_freq[:,:,0]), ref=np.max), sr=sample_rate, hop_length=
hop_size, x_axis='time', y_axis='log',cmap='jet')
plt.title('Original noisy Speech in dB')
```

```
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.colorbar(format='%+2.0f dB')

plt.subplot(2,1,2)
librosa.display.specshow(librosa.amplitude_to_db(abs(enhanced_speech_freq), ref=np.max), sr=sample_rate, hop_length=hop_size, x_axis='time', y_axis='log',cmap='jet')
plt.title('Enhanced Speech in dB by MMSE in multi-channel')
plt.xlabel('Time')
plt.ylabel('Frequency')
plt.colorbar(format='%+2.0f dB')
plt.subplots_adjust(hspace=0.5, wspace=0.5)
plt.show()

sf.write("enhce_mmse.wav", enhanced_speech, fs)



#Evaluion for 0-16 channels by SNR and var_emp
enhanced_speech_all_mmse=np.zeros((len(enhanced_speech),16))
enhanced_speech_freq_all_mmse=np.zeros((321,3761,16))
snr_all_mmse=np.zeros(16)

for i in range(16):
    enhanced_speech_all_mmse[:,i], enhanced_speech_freq_all_mmse[:,:,i] = MMSE_MultiChannel_speech_enhancement(speech_noisy_freq[:,:,0:i+1])
    snr_all_mmse[i] = SNR(enhanced_speech_all_mmse[:,i])

#plot the enhanced speech for all 16 channels
#subplots
plt.figure(figsize=(45,25))
for i in range(16):
    plt.subplot(4,4,i+1)
    librosa.display.specshow(librosa.amplitude_to_db(abs(enhanced_speech_freq_all_mmse[:,:,i]), ref=np.max), sr=sample_rate, hop_length=hop_size, x_axis='time', y_axis='log',cmap='jet')
    plt.title(str(i+1)+' Channels')
    plt.xlabel('Time')
    plt.ylabel('Frequency')
    plt.colorbar(format='%+2.0f dB')


#plot snr for each channel
print("SNR of the enhanced speech for all 0-16 channels is: ", snr_all_mmse)
plt.figure(figsize=(10,5))
plt.scatter(range(1, 17), snr_all_mean)
plt.scatter(range(1, 17), snr_all_mle)
plt.scatter(range(1, 17), snr_all_mmse)

plt.title('SNR for 0-16 channels after multi enhancement by MMSE')
plt.xticks(range(1, 17))
plt.xlabel('Number of Channels')
plt.ylabel('SNR')
plt.grid()
#legand
plt.legend(['Enhancement by averaging the channels','Enhancement by MLE', 'Enhancement by MMSE'])
plt.show()


# calculate the Var_emp
var_emp_all_mmse=Var_emp(enhanced_speech_freq_all_mmse)
var_emp_all_mean=Var_emp(speech_enhanced_freq_all_mean)
print("var_emp of the enhanced speech for all 0-16 channels is: ", var_emp_all_mmse)
plt.figure(figsize=(10,5))
plt.scatter(range(1, 17), var_emp_all_mean)
plt.scatter(range(1, 17), var_emp_all_mle)
plt.scatter(range(1, 17), var_emp_all_mmse)
plt.plot(range(1, 17), CRLB, color='red')
plt.title('var_emp for 0-16 channels after multi enhancement by MMSE')
plt.xticks(range(1, 17))
plt.xlabel('Number of Channels')
plt.ylabel('var_emp')
plt.legend(['Enhancement by averaging the channels','Enhancement by MLE', 'Enhancement by MMSE','CRLB'])
plt.grid()
plt.show()

# calculate the Var_emp with only MMSE
plt.figure(figsize=(10,5))
plt.scatter(range(1, 17), var_emp_all_mmse, color='green')
plt.plot(range(1, 17), CRLB, color='red')
plt.title('var_emp for 0-16 channels after multi enhancement by MMSE')
plt.xticks(range(1, 17))
```

```
plt.xlabel('Number of Channels')
plt.ylabel('var_emp')
plt.legend(['Enhancement by MMSE','CRLB'])
plt.grid()
plt.show()



#------------------END----------------------#
```