Niharika V
2018122008

Speech Signal Processing
Assignment 3

Q1. 2D & 3D STFT Plot for Speech Signal :

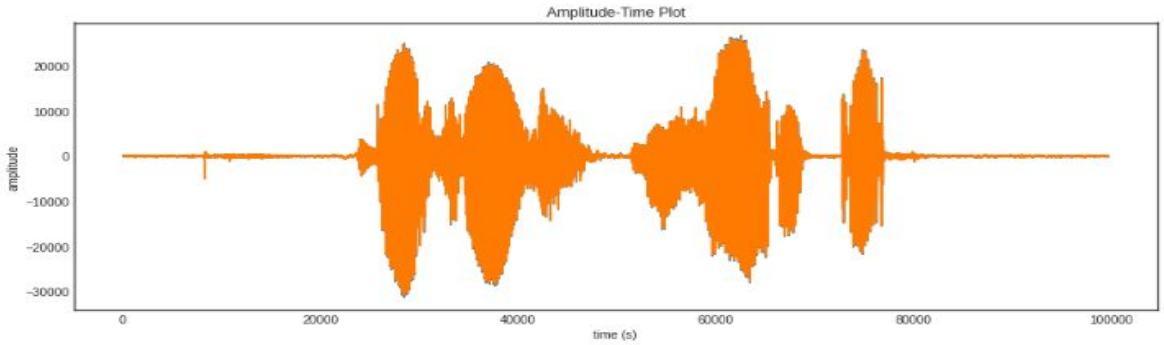
A1 : - We need to perform STFT on speech signals , as they tend to be non-stationary in nature , and hence frequency varies depending on type of phone production . Therefore for speech we need a representation that will give time varying spectral information.The entire speech signal and gives us only gross information of the frequency components present in the speech signal. However, this spectrum does not tell us when a particular frequency component is present i.e the timing information is completely lost.

- Python Code Snippet :

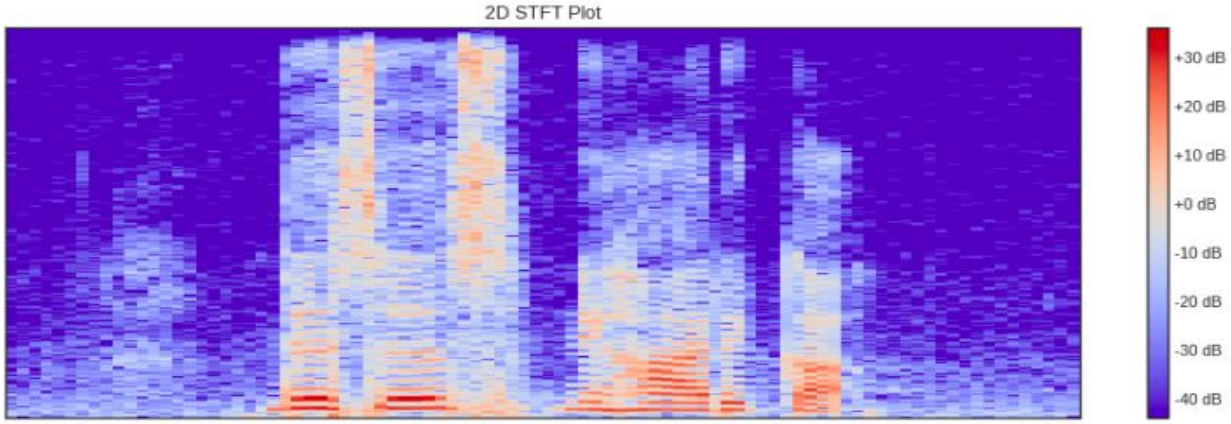
```
def stftPlot(fileName,nfft=512,hopLength=512,choice='2D',plot=True):
    # Read the speech sample .
    data,Fs= librosa.load(str(fileName))
    if(data.shape[0]==2):
        data=float(data[:,0])
    time =np.linspace(0,len(data),len(data))
    # Stft
    X = librosa.stft(data, n_fft=nfft, hop_length=hopLength)
    X_mag=librosa.amplitude_to_db(abs(X))
    S = librosa.amplitude_to_db(abs(X_mag))

    if(choice=='2D'):
        plt.figure(figsize=(15, 5))
        librosa.display.specshow(X_mag,sr=Fs, hop_length=hopLength)
        plt.title('2D STFT Plot')
        plt.colorbar(format='%+2.0f dB')

    if(choice=='3D'):
        fig=plt.figure(figsize=(15, 5))
        ax = fig.gca(projection='3d')
        t,f,spec=signal.stft(data,Fs,nfft=512,nperseg=512)
        T,F = np.meshgrid(t,f)
        Sx=np.transpose(10.0*np.log10(abs(spec)))
        surf=ax.plot_surface(T,F,Sx,cmap='viridis')
        fig.colorbar(surf, format='%+2.0f dB')
        plt.show()
```

- Speech Sample Time-Amplitude Plot :

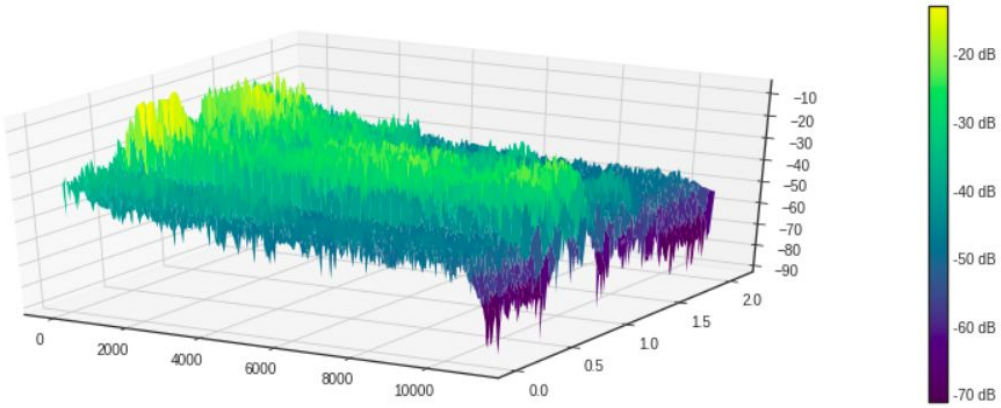
```
File Statistics
File Name :  /data/niharik
Number of Channels : 2
Sample Rate :  48000 Hz
Time Duration:  2.0756666666666668 secs
```



- 2D Spectrogram Plot :



- 3D Spectrogram Plot :



Q 2 .Load a speech signal of your interest and select the voiced region then Find the pitch based on the Cepstral analysis? Plot the results (Speech, Cepstrum, Liftered signal).

A2 :

- **Cepstral Analysis :-**

According to the source filter theory of speech production, voiced sounds are produced by exciting the time varying system characteristics with periodic impulse sequence and unvoiced sounds are produced by exciting the time varying system with a random noise sequence. From the signal processing point of view, the output of a system can be treated as the convolution of the input excitation with the system response.

The resulting speech can be considered as the convolution of respective excitation sequence and vocal tract filter characteristics. If e(n) is the excitation sequence and h(n) is the vocal tract filter sequence, then the speech sequence s(n) can be expressed as follows:

$$s(n) = e(n) * h(n)$$

$$|S(\omega)| = |E(\omega)|.|H(\omega)|$$

For this, multiplication of the two components in the frequency domain has to be converted to a linear combination of the two components. For this purpose cepstral analysis is used for transforming the multiplied source and system components in the frequency domain to linear combination of the two components in the cepstral domain.

$$log|S(\omega)| = log|E(\omega)| + log|H(\omega)|$$

Here, c(n) is the cepstral coefficient analysis .

$$c(n) = IDFT(log|S(\omega)|) = IDFT(log|E(\omega)| + log|H(\omega)|)$$

- Block Diagram

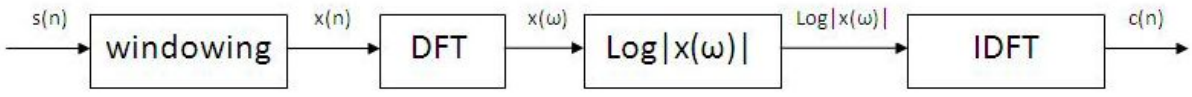


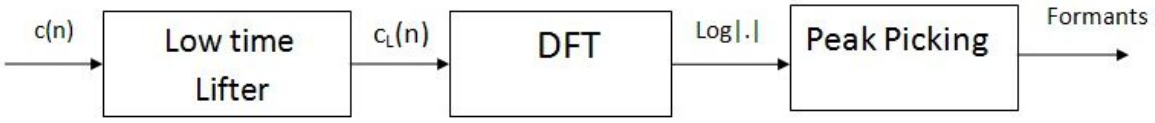
Figure 1: Block diagram representing computation of cepstrum

- Liftering Concept :

  Liftering operation is similar to filtering operation in the frequency domain where a desired quefrency region for analysis is selected by multiplying the whole cepstrum by a rectangular window at the desired position. There are two types of liftering performed, low-time liftering and high-time liftering. Low-time liftering operation is performed to extract the vocal tract characteristics in the quefrency domain and high-time liftering is performed to get the excitation characteristics of the analysis speech frame. Here, ce(n) will be the liftered signal .

$$w_e[n] = \begin{cases} 1, 0 \le n \le L_c \\ 0, L_c \le n \le \frac{N}{2} \end{cases} \qquad c_e(n) = w_e[n].\, c(n)$$

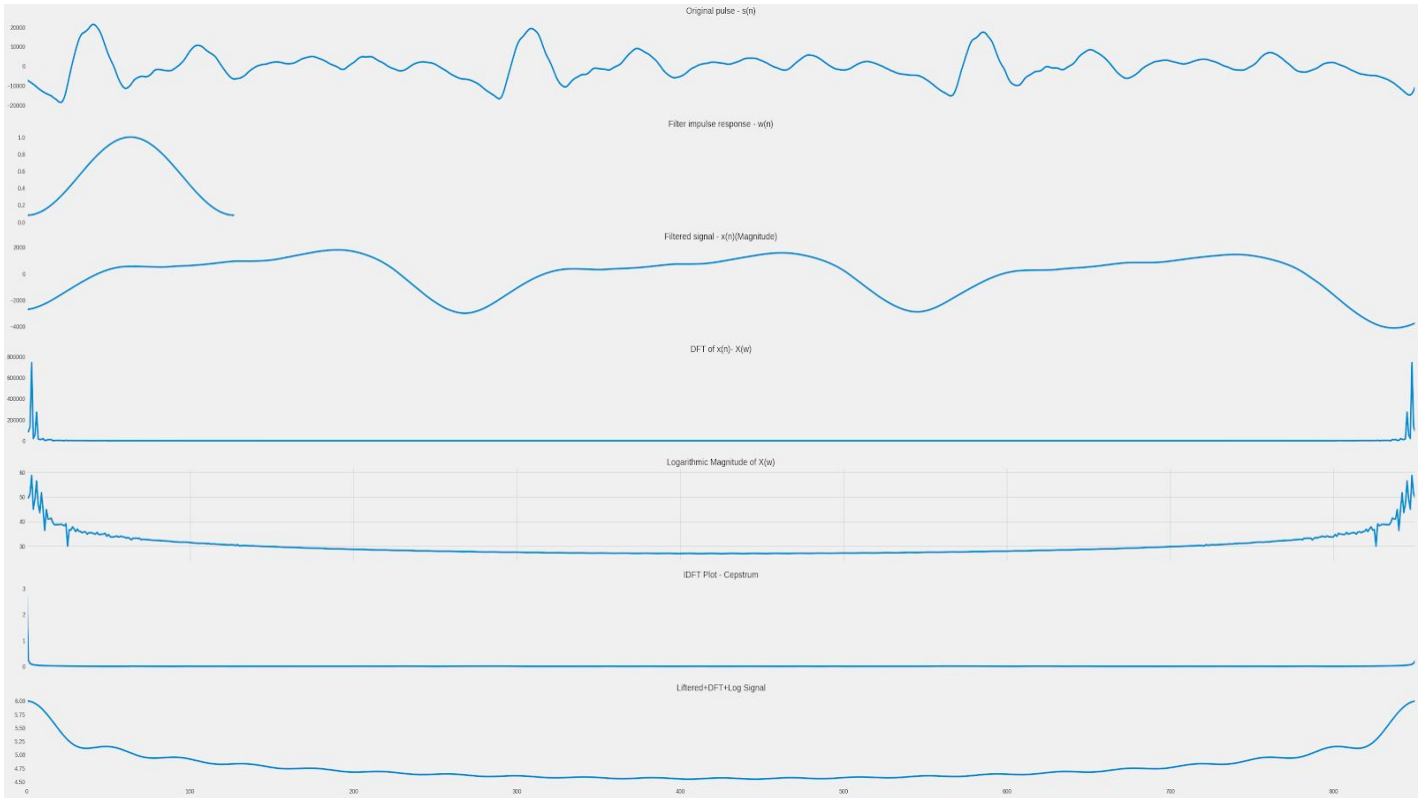
  The entire block for format extraction will be : :



  - Python Code :

```
def cepstralAnalysis(fileName,N=128,Lc=20,plot=False):
    Fs,data=wavfile.read(str(fileName))
    data=data[:, 0].flatten()
    # Apply Hamming Window.
    window=signal.hamming(N)
    filtered = np.convolve(data,window, 'same')/sum(window)
    # DFT
    x_fft=np.fft.fft(filtered)
    # Log DFT
    x_fft_log=10*np.log10(abs(x_fft))
    # Inverse DFT - Cepstrum .
    x_cepstrum=np.fft.ifft(0.1*x_fft_log)
    # Liftered Signal
    Nl=len(x_cepstrum)
    Lc=20
    low_lifter=[1 if(i>=0 and i<=Lc) else 0  for i in range(0,int(Nl),1)]
    x_liftered=10*np.log10(abs(np.fft.fft(np.multiply(low_lifter,x_cepstrum))))
```

- Output :



Comments :

- Since we need major formants of the voiced speech , I've applied a low-lifter . Also,I've set Lc = 15 , in this case and N = total length of cepstrum , and performing low -time liftering for formants since we are working with voiced signals. After applying a lifter to the cepstrum , we can observe one sharp peak - which is the formant .

Q3 : Consider a voiced frame and unvoiced frame of your choice, apply the following.
(a) no. of points in N-point DFT (256,512,1024)
(b) size of window
(c) shape of window (Rectangular, Hamming, Hanning) on STFT

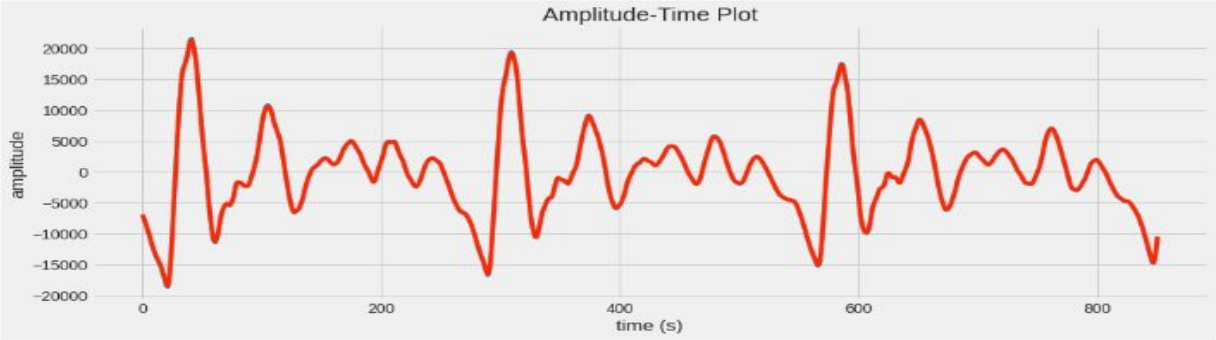
NOTE : In STFT, there are many frames when you plot log-magnitude spectrum across time , I've taken the first frame to clearly observe the effect of each parameter . In case you wish to see across frame , do the following :
-- Change :
plt.plot(fscale,10*np.log10(abs[Sx[:,0]))
--To
plt.plot(fscale,10*np.log10(abs[Sx]))

A3 **c) Shape of Window** : For the voiced part , I've taken a small sample of vowel - 'a' and for an unvoiced part - a consonant 'k' . These files are in /data folder with the names - a_frame.wav and k_frame.wav.
Note : The size of window - N = 256 and other parameters are fixed.
Python Code Snippet :

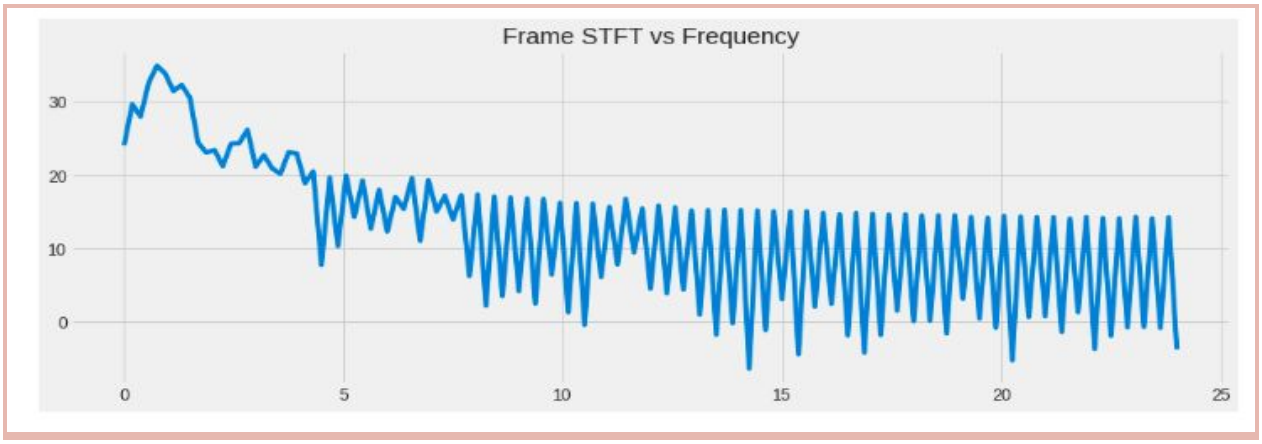
```
def stftWindow(fileName,win='hann',N=256,plot=True):
    Fs,data=wavfile.read(str(fileName))
    data=data[:, 0].flatten()
    print(data.shape)
    freq,time,Sx = signal.stft(data,Fs, window=win, nperseg=256,nfft=N,return_onesided=True)
    # Lets plot the log Magnitude Specturum .

    if(plot):
        fig=plt.figure(figsize=(15, 5))
        plt.style.use('fivethirtyeight')
        fscale=np.linspace(1/Fs,Fs/2000,len(list(Sx[:,0])))
        plt.plot(fscale,10.0*np.log10(abs(Sx[:,0])))
        plt.title('Frame STFT vs Frequency')
```

Voice Speech Sample :

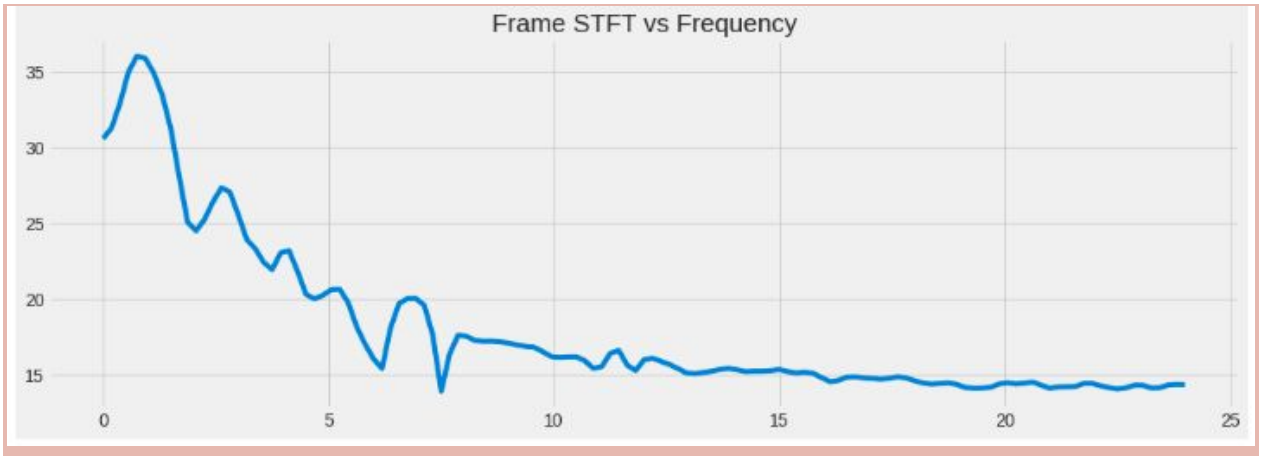
```
File Statistics
File Name :  /data/a_frame
Number of Channels : 2
Sample Rate :  48000 Hz
Time Duration:  0.017729166666666667 secs
```



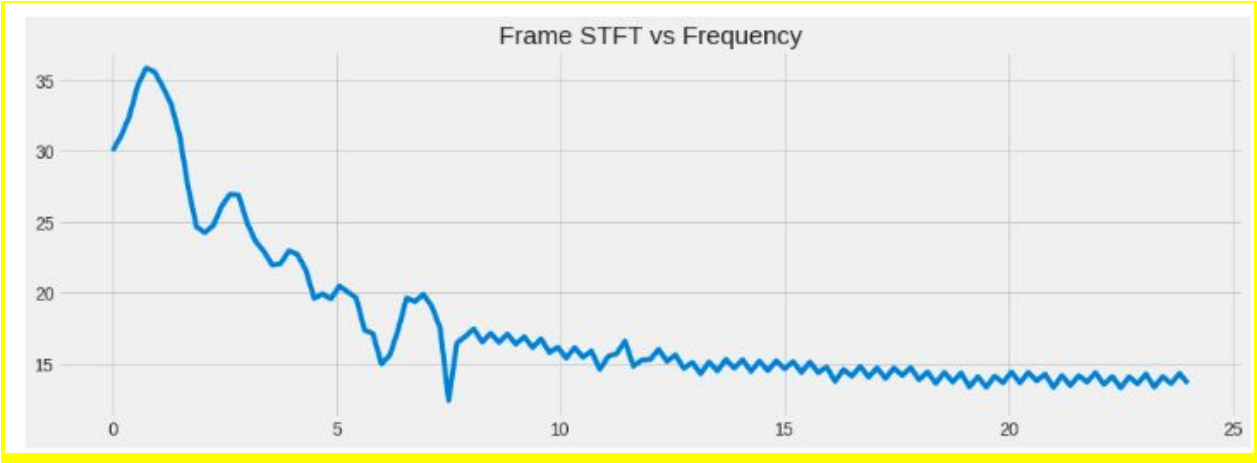
Rectangular Window :



Hanning Window :

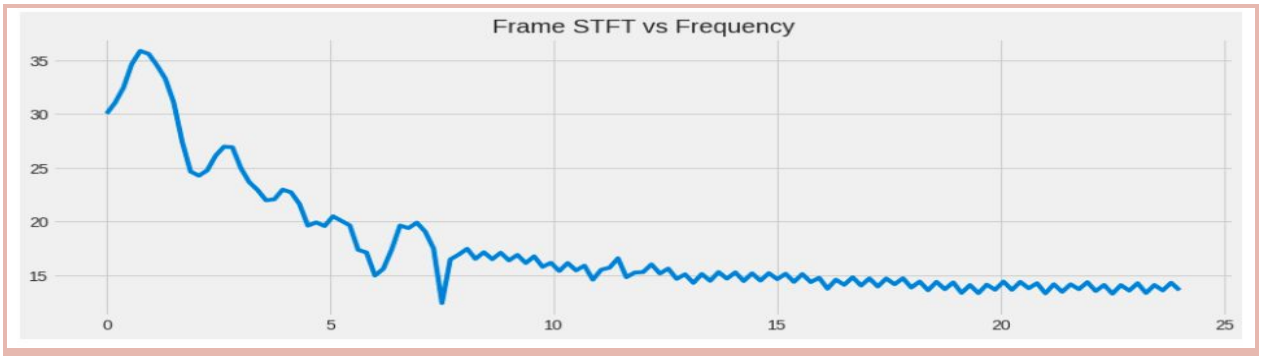


Hamming Window :

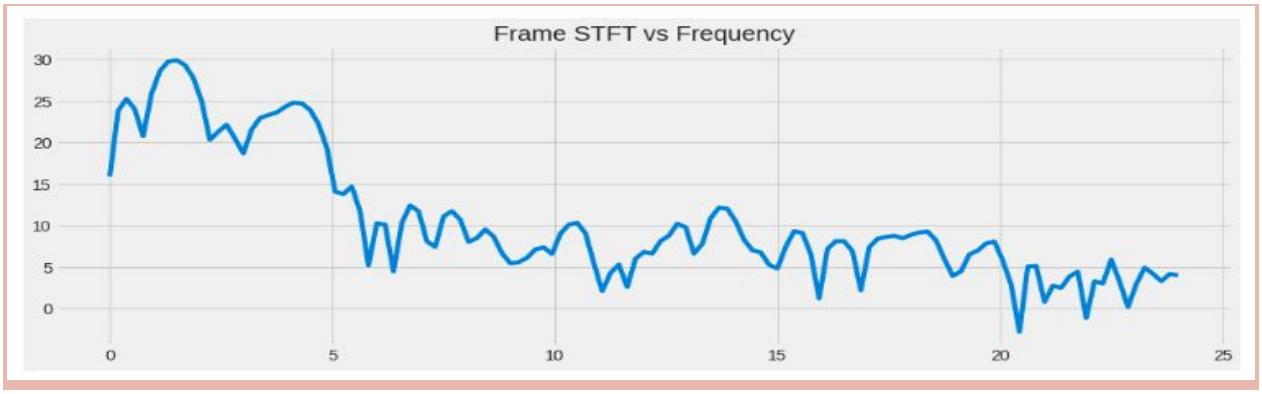


- **Case : Unvoiced Sample**

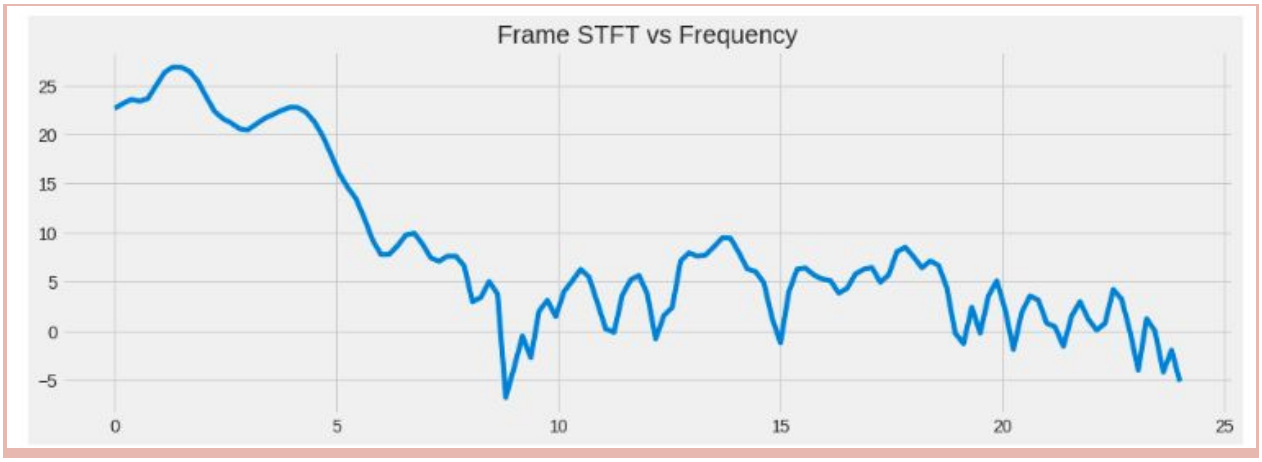
Hamming Window



Rectangular Window :



Hanning Window :



Remarks:

- The peaks in the log magnitude spectrum of the rectangular window (especially voiced segment) are relatively sharper compared to the other cases.
- However, the spectrum due to the rectangular window seems to be more noisy compared to the other two window functions. This is because of the high spectral leakage associated with it.
- **Spectral leakage** occurs when a non-integer number of periods of a signal is sent to the DFT.,as it lets a single-tone signal spread among several frequencies after the DFT operation. This makes it hard to find the actual frequency of the signal.
- Similar observations can be made with respect to the unvoiced segment of speech shown . Since spectral leakage is severe in case of rectangular window, either Hamming or Hanning window is preferred for the short term spectral analysis of speech.
- The main difference between them is that the Hanning window touches zero at both ends, removing any discontinuity. The Hamming window stops just shy of zero, meaning that the signal will still have a slight discontinuity . In our case , we can also observe that in all the Hamming window plots , there are small several peaks

A3 b) Number of points in N-DFT

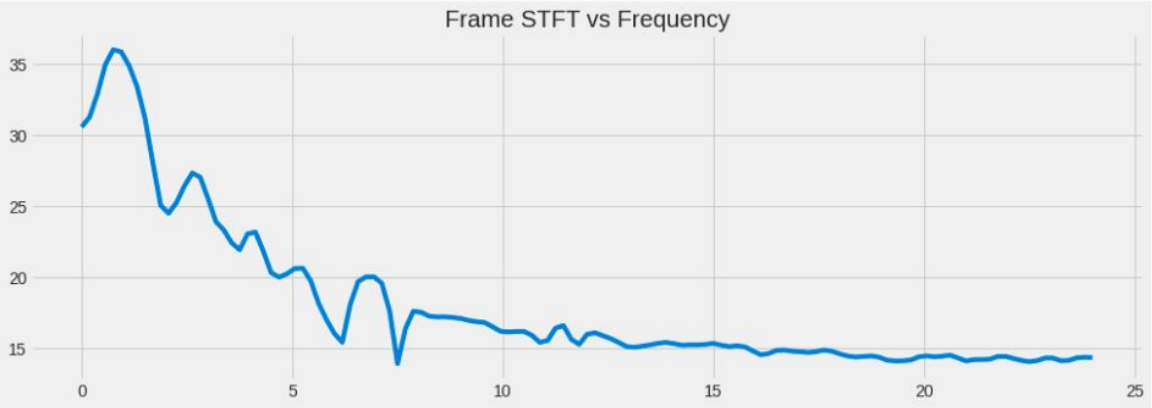
- The experiment is repeated on the same voice samples as above , and we are experimenting with N= 256 , 512,1024 , and we set other parameters to default - hanning window,nperseg=256 .

Python Code

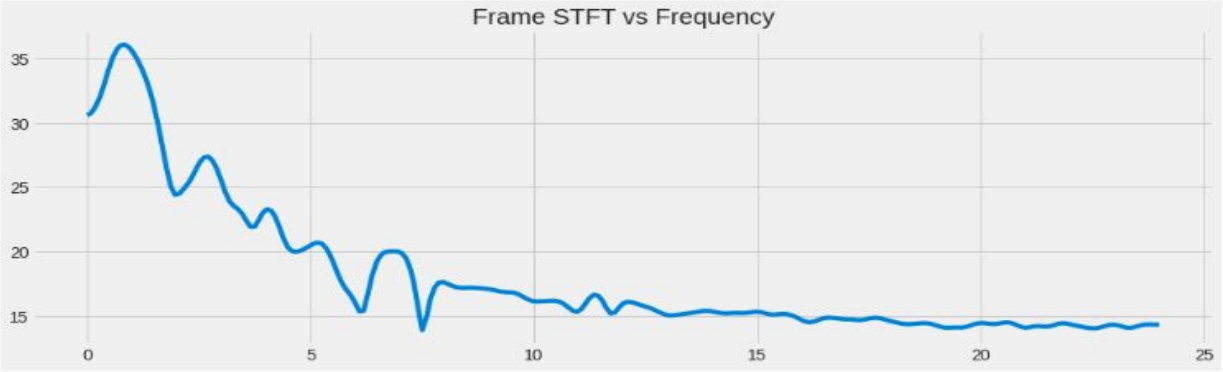
```
def stftNpoints(fileName,win='hann',Nfft=256,plot=True):
    Fs,data=wavfile.read(str(fileName))
    data=data[:, 0].flatten()
    print(data.shape)
    freq,time,Sx = signal.stft(data,Fs, window=win,nperseg=256,nfft=Nfft,return_onesided=True)
    # Lets plot the log Magnitude Specturum .

    if(plot):
        fig=plt.figure(figsize=(15, 5))
        plt.style.use('fivethirtyeight')
        fscale=np.linspace(1/Fs,Fs/2000,len(list(Sx[:,0])))
        plt.plot(fscale,10.0*np.log10(abs(Sx[:,0])))
        plt.title('Frame STFT vs Frequency')
```

Case : Voiced Speech Sample

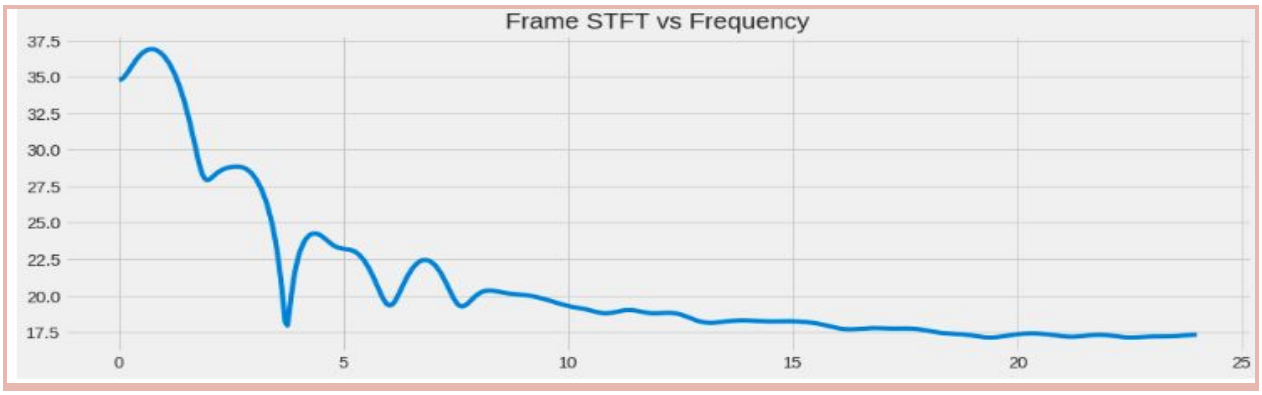
- N= 256



- N= 512

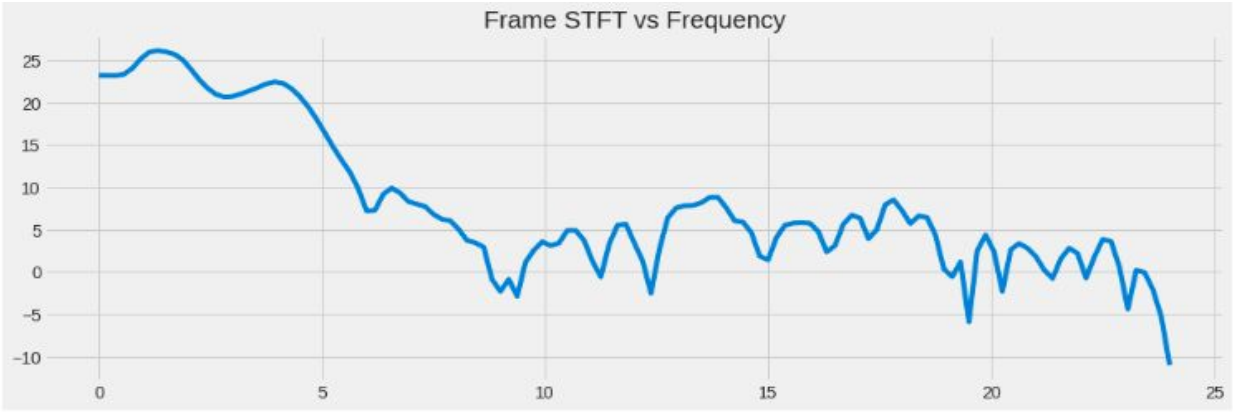


- N=1024

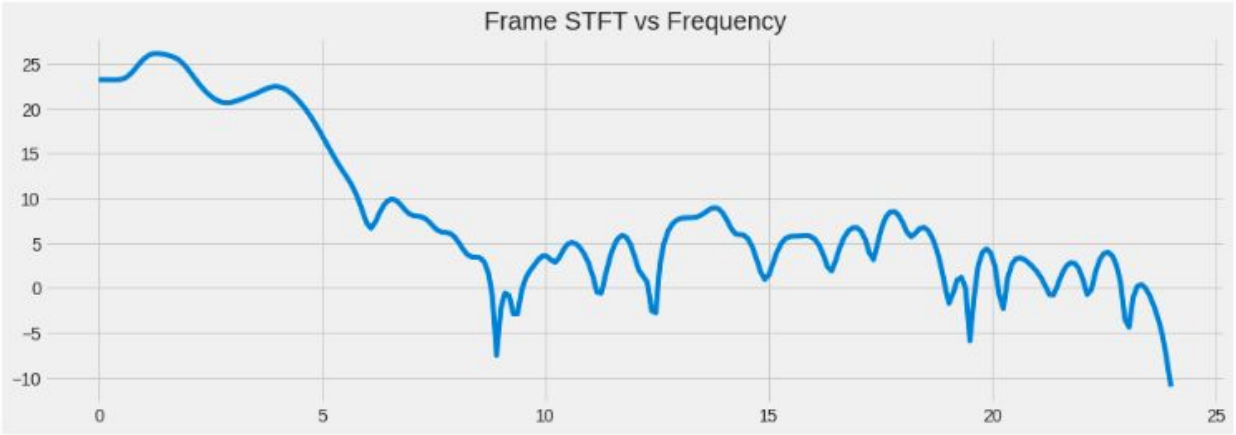


Case : Unvoiced Sample

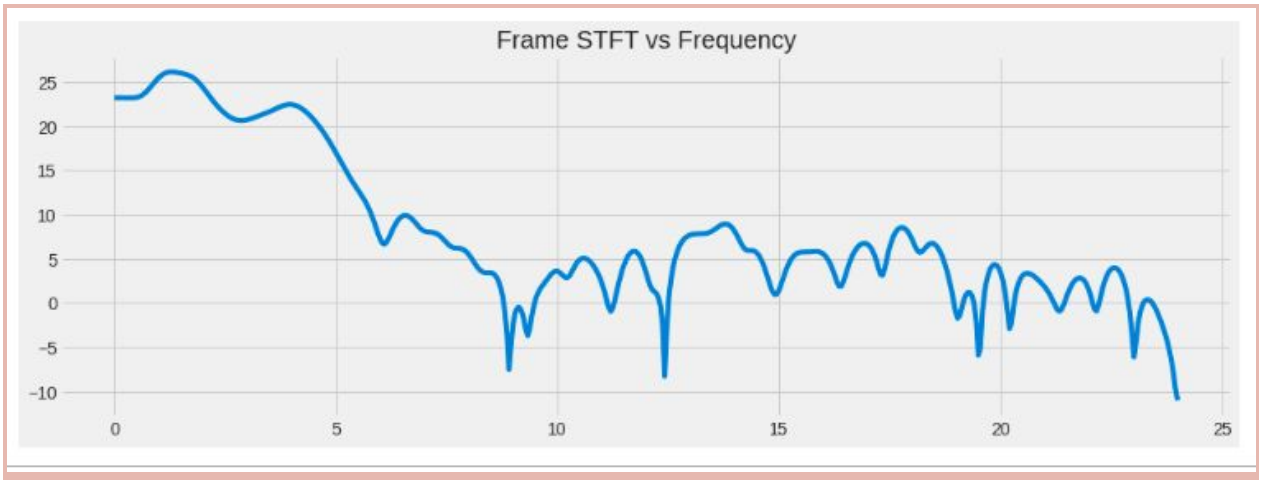
- N = 256



- N= 512



- N=1024



Remarks :

- The length N of the DFT is the number of frequency points that will result in the DFT output. Zero padding will result in more frequency samples, however this does not increase frequency resolution, it just interpolates samples in the DTFT.
- The frequency resolution is given by 1/T where T is the time length of your data (regardless of sampling rate). So if you want to increase the actual frequency resolution, you need to increase the number of samples at a given sampling rate, or decrease the sampling rate which would increase the time length for the number of samples you have.
- Hence , here when we are increasing (N) parameter - increases the number of samples while computing the FFT , the frequency resolution improves .
- In both voiced and unvoiced cases , you can see how the resolution of the graph has increased . (Note: Need to look carefully, but you can see peaks are sharpening and new peaks have formed) .

c) Effect of Window Length in STFT:

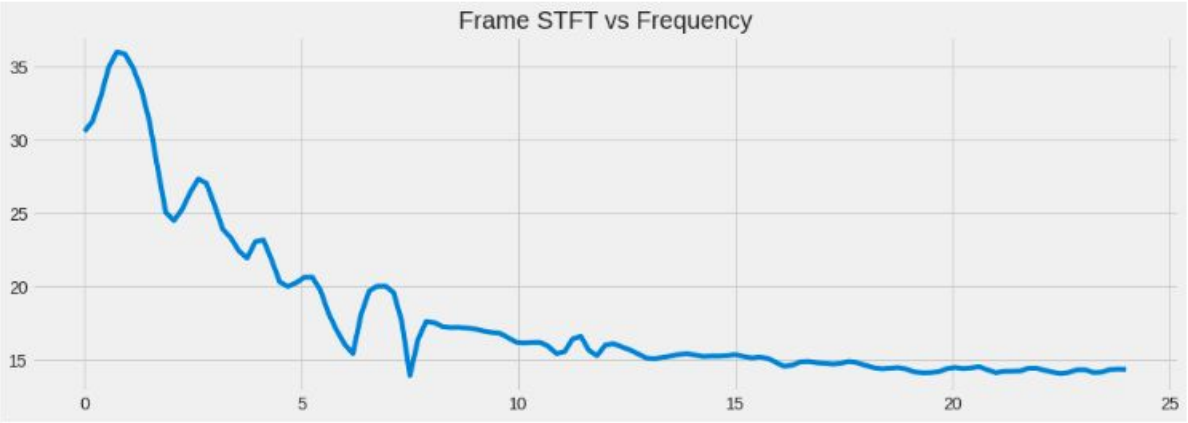
- The experiment is repeated on the same voice samples as above , and we are experimenting with N= 256 , 512,1024 , and we set other parameters to default - hanning window
  & here I'm setting **Nfft = Nperseg (as Nfft should be greater than or equal to Nperseg - Python)**
- Window length here refers to STFT's frame size . Here , I have gradually increased from- 128,512,1024 .

Python Code Snippet :

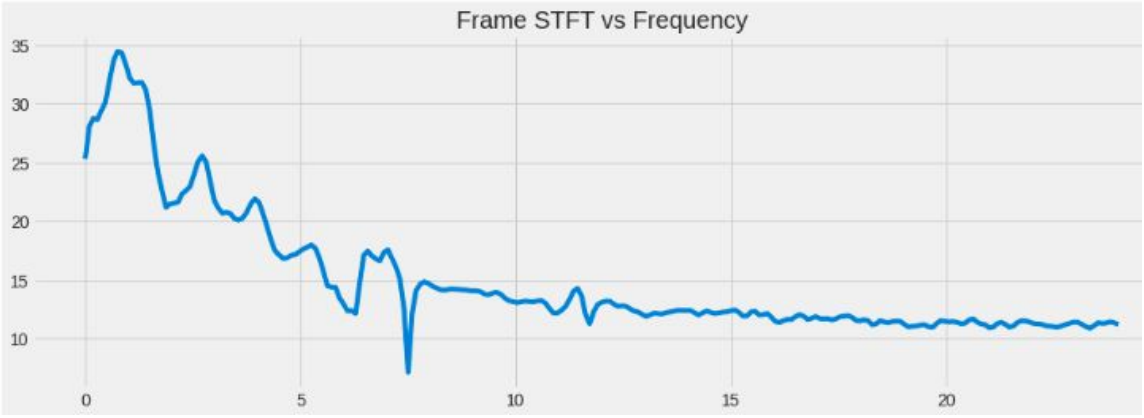
```
def stftWindowLength(fileName,Nper=128,plot=True):
    Fs,data=wavfile.read(str(fileName))
    data=data[:, 0].flatten()
    print(data.shape)
    freq,time,Sx = signal.stft(data,Fs, window='hann',nperseg=Nper,nfft=128,return_onesided=True)
    # Lets plot the log Magnitude Specturum .
    if(plot):
        fig=plt.figure(figsize=(15, 5))
        plt.style.use('fivethirtyeight')
        fscale=np.linspace(1/Fs,Fs/2000,len(list(Sx[:,0])))
        plt.plot(fscale,10.0*np.log10(abs(Sx[:,0])))
        plt.title('Frame STFT vs Frequency')
```

Case 1 : Voiced Sample

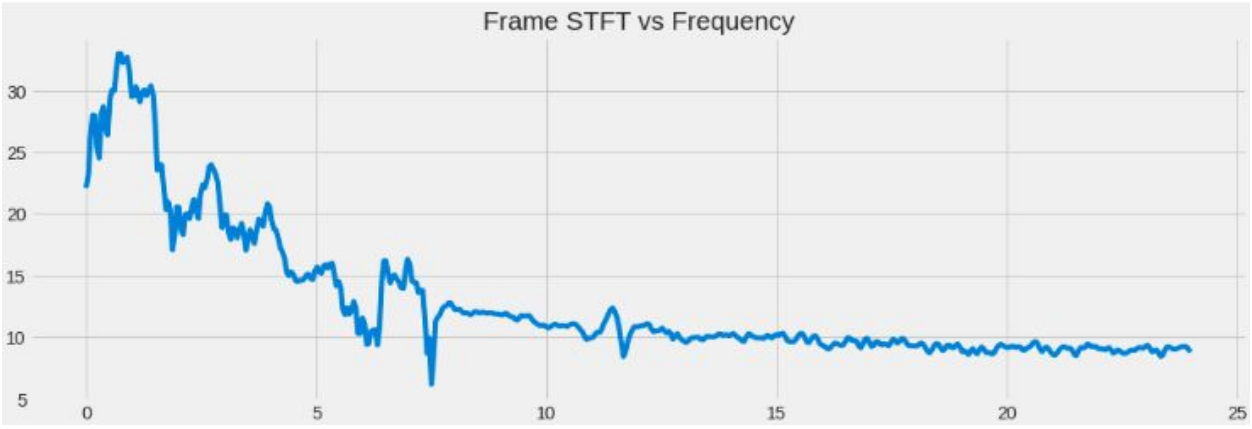
- Window Length =256



- Window Length =512

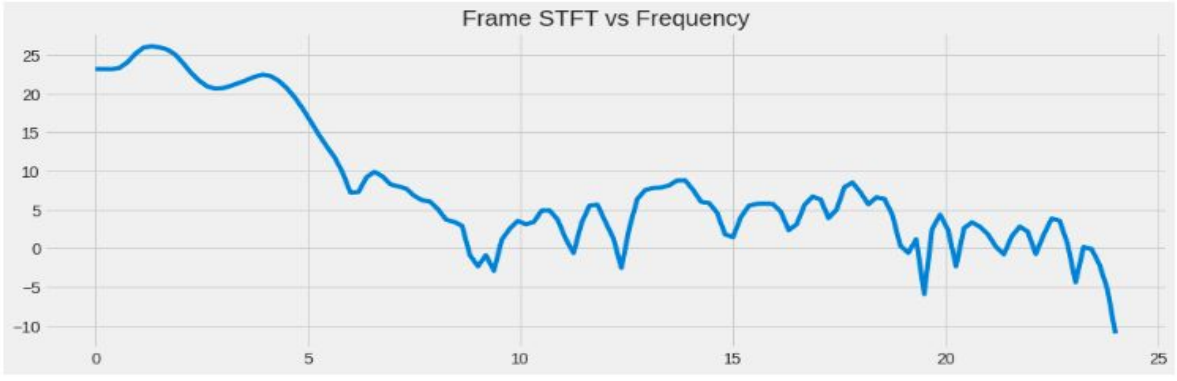


- Window Length = 1024

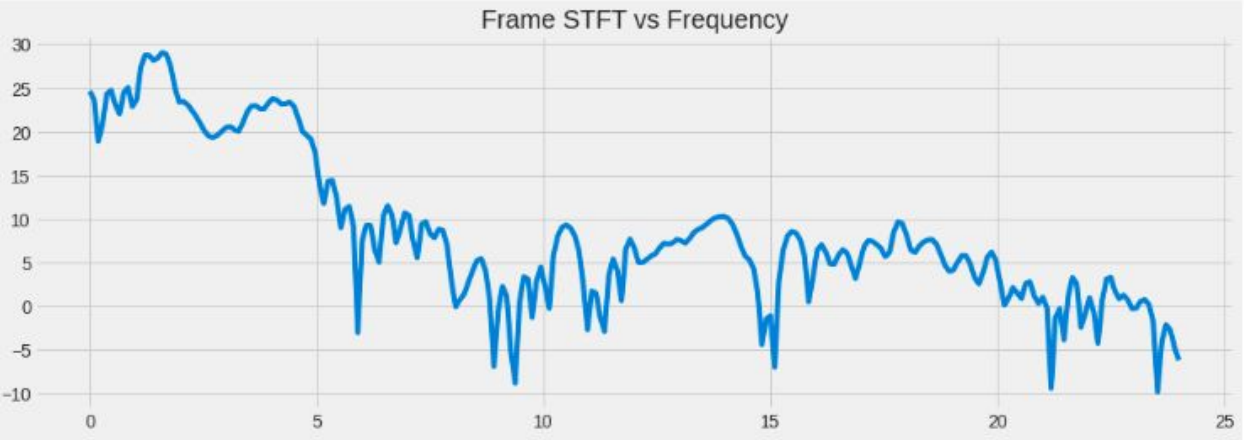


Case 2 : Unvoiced Sample

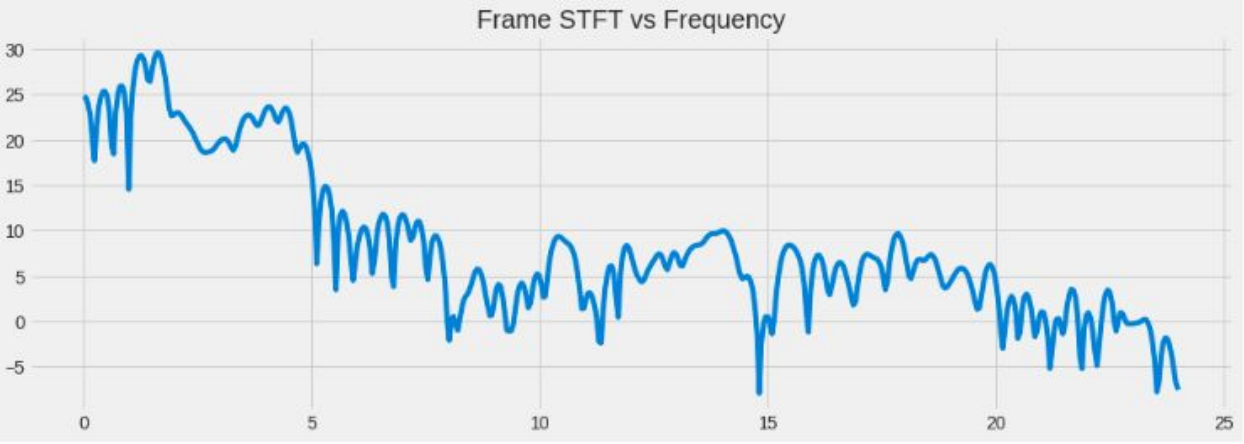
- Window Length =256



- Window Length =512



- Window Length = 1024



- **Remarks :**

  - In the smallest window case , since the window size is less than a pitch period, the log magnitude spectrum has only a smooth envelope representing vocal tract information. However, due very short frame size, the vocal tract information is affected from poor spectral resolution.
  - Finally, in the longer frame size (1024 samples case) case, the spectrum suffers from timing resolution due to its large size. Similar observations can be made for unvoiced segmentation of speech also.
  - We can see that Nperseg=1024 cases have most peaks and since the default is set to 'hanning', it shows characteristics of that window , which is absent in Nperseg=256 case .

- We can minimize this effect by using a proper size of window. If the window duration is of 10-30 ms, then the spectral information may be affected to a minimum extent.
- Alternatively, if the window size is too less, then the window effect may be too severe. In yet another case, if the window is too large , say 100 msec, then even though the windowing effect is negligible compared to the 10-30 ms case, we cannot use such a long window due to the non-stationary nature of speech.

- Note : Mathematically , we can say that properties of this Fourier transform depend on the window frequency resolution with the length of the window .Hence, we require long windows for high resolution , but only in case of relatively stationary signals during window duration .

- But, since speech is quasi-stationary , we need to compromise between good temporal resolution(short windows) and good frequency resolution (long windows).

- Here , Nperseg = 256 seems to be a balanced choice , across both voiced and unvoiced parts .

Q3 . Take a recorded speech signal and extract MFCC for every 20 ms and save it in txt file. Plot accordingly as shown below Figure. 3.

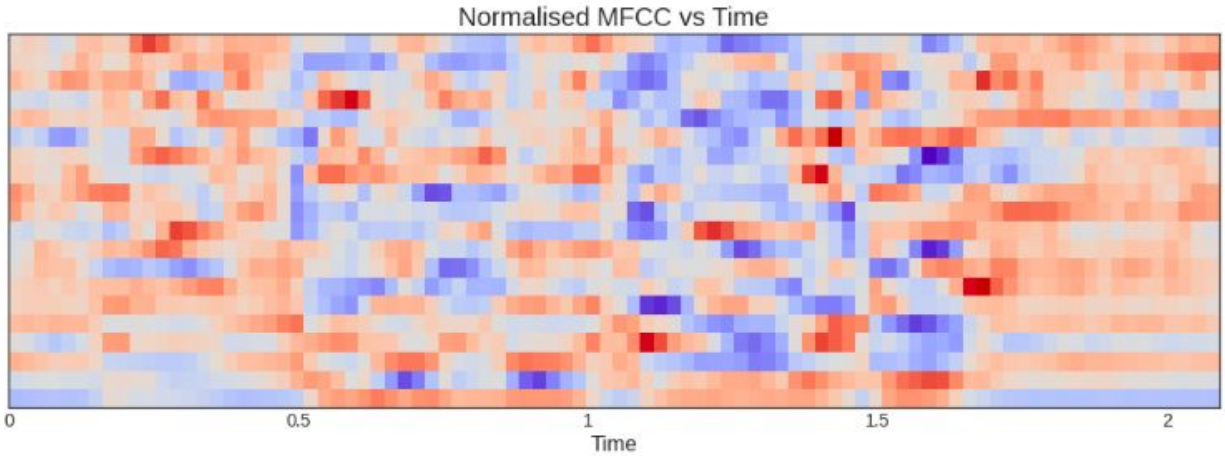
A3 : MFCC — Mel-Frequency Cepstral Coefficient : This feature is one of the most important methods to extract a feature of an audio signal and is used majorly whenever working on audio signals. The mel frequency cepstral coefficients (MFCCs) of a signal are a small set of features (usually about 10–20) which concisely describe the overall shape of a spectral envelope.A MEL scale is a unit of PITCH proposed by Stevens, Volkmann and Newmann in 1937. The MEL scale is a scale of pitches judged by listeners to be equal in distance one from another . Because of how humans perceive sound the MEL scale is a non-linear scale and the distances between the pitches increases with frequency .

Part 1 : MFCC vs Time

Code Snippet :

```
def MFCCTime(fileName,plot=True):
    x, fs = librosa.load(fileName)
    mfccs = librosa.feature.mfcc(x, sr=fs)
    print('MFCC shape : ',mfccs.shape)
    if(plot):
        fig=plt.figure(figsize=[15,5])
        mfccs = sklearn.preprocessing.scale(mfccs, axis=1)
        surf=librosa.display.specshow(mfccs, sr=fs, x_axis='time')
        plt.title('Normalised MFCC vs Time')
        plt.show()
```

Plot : (Normalised the MFCC)



Part 1 : MFCC vs Frame Index Part

```
# MFCC-gram (i.e across the signal)
def MFCCFrame(fileName,Nc=13,frameSize=512,nperseg=256,plot=True):
    data,fs=librosa.load(fileName)
    hamming_window = ess.Windowing(type='hamming')
    spectrum = ess.Spectrum()  # we just want the magnitude spectrum
    mfcc = ess.MFCC(numberCoefficients=13)
    mfccs = np.array([mfcc(spectrum(hamming_window(frame)))[1]
                      for frame in ess.FrameGenerator(data, frameSize=frameSize, hopSize=nperseg)])
    mfccs = sklearn.preprocessing.scale(mfccs)
    plt.imshow(mfccs.T, origin='lower', aspect='auto', interpolation='nearest',cmap='viridis')
    plt.ylabel('MFCC Coefficient Index')
    plt.xlabel('Frame Index')
    print(mfccs.shape)
```

Plot :