18D070067_Assignment2

October 18, 2021

ASSIGNMENT 2 EE679

Name: VINIT AWALE

Roll No. : 18D070067

Question

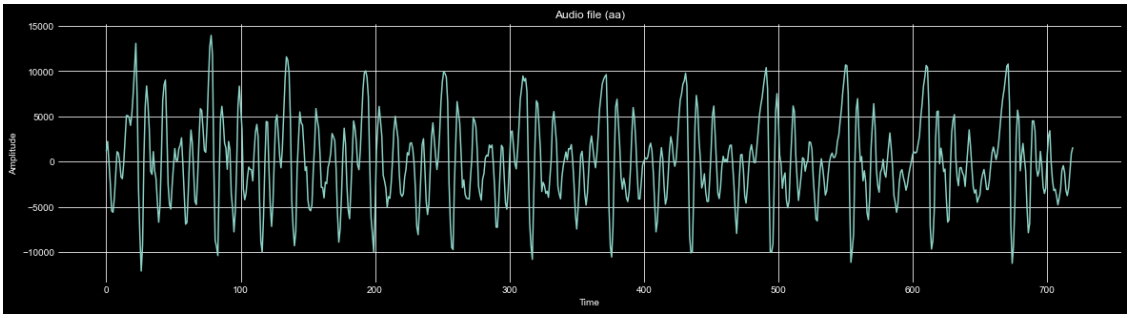
Given the speech segment (aa.wav) extracted from the word “pani” in “machali.wav” (male voice), sampled at 8 kHz, do the following. Report/discuss your observations at each step.

IMPORTS

[ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import IPython
```

[ ]:
```
## Let us first read the given audio file
import scipy.io.wavfile as wav

sampling_rate, aa = wav.read('aa.wav')
```

[ ]:
```
## Visualize the audio file
plt.figure(figsize=(20,5))
plt.plot(aa)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('Audio file (aa)')
plt.show()
```



[ ]:
```
sampling_rate
```

[ ]:
```
8000
```

Hence, as mentioned in the question the sampling rate of the audio file is 8 kHz

1 Part 1

1.1 Apply pre-emphasis to the signal.

Consider the following pre-emphasis filter

$$H(z) = 1 - \alpha z^{-1}$$

where α is a constant. For this assignment let α be 0.95. Hence, the filter is given by

$$H(z) = 1 - 0.95z^{-1}$$
$$\implies \frac{Y(z)}{X(z)} = 1 - 0.95z^{-1}$$

Where, Y(z) is the z-transform of the output signal and X(z) is the z-transform of the input signal to the filter.

$$\implies Y(z) = X(z) - 0.95X(z)z^{-1}$$

Hence, after taking inverse z transform, we get

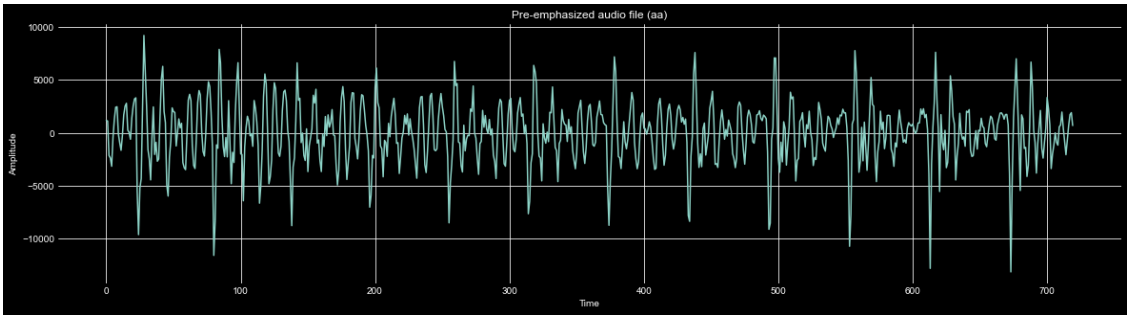
$$\implies y[n] = x[n] - 0.95x[n-1]$$

Now, let us find the output when the "aa" signal is passed through the pre-emphasis filter.

[ ]:
```
pre_emphasis_aa = np.zeros(len(aa))

pre_emphasis_aa[0] = aa[0]

for i in range(1, len(aa)):
    pre_emphasis_aa[i] = aa[i] - 0.95 * aa[i-1]
```

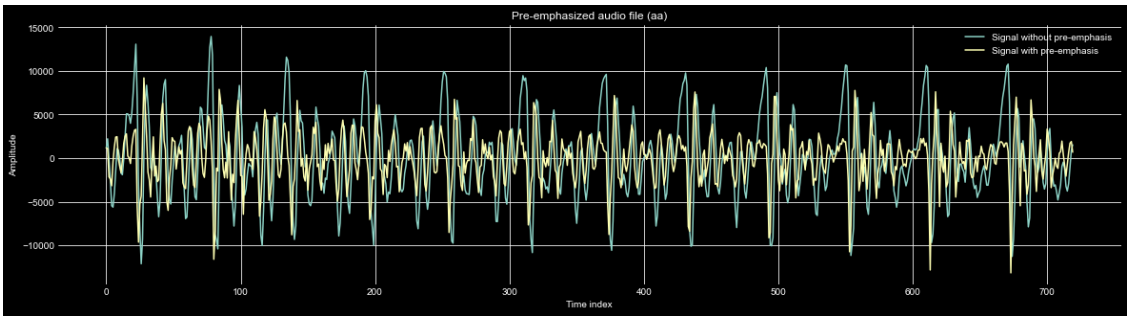
1.1.1 Visualization of the pre-emphasized signal

[ ]:
```
plt.figure(figsize=(20,5))
plt.plot(pre_emphasis_aa)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('Pre-emphasized audio file (aa)')
plt.show()
```



1.1.2 Combined Plot

[ ]:
```
plt.figure(figsize=(20,5))
plt.plot(aa , label = "Signal without pre-emphasis")
plt.plot(pre_emphasis_aa, label = "Signal with pre-emphasis")
plt.xlabel('Time index')
plt.ylabel('Amplitude')
plt.title('Pre-emphasized audio file (aa)')
plt.legend()
plt.show()
```



1.1.3 Writing the output of pre_emphasis_aa to a file

[ ]:
```
from scipy.io.wavfile import write

pre_emphasis_aa_sound = np.int16(pre_emphasis_aa/np.max(np.
 ↪abs(pre_emphasis_aa)) * 32767)
write('pre_emphasis_aa.wav', sampling_rate, pre_emphasis_aa_sound)
```

2 Part 2

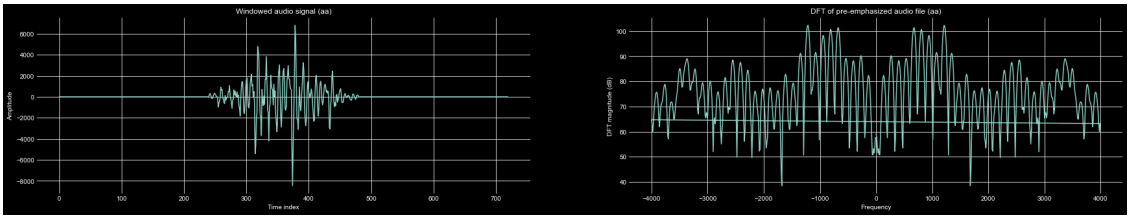
2.1 Compute and plot the narrowband magnitude spectrum slice using a Hamming window of duration = 30 ms on a segment near the centre of the given audio file.

[ ]:
```
## Hamming window of duration 30 ms
hamming_window = np.hamming(30e-3*sampling_rate)

## Append zeros on both sides of Hamming window to make it of the same lenght␣
 ↪as the pre-emphasized audio file
hamming_window = np.pad(hamming_window,␣
 ↪((len(pre_emphasis_aa)-len(hamming_window))//2 ,␣
 ↪(len(pre_emphasis_aa)-len(hamming_window))//2), 'constant')

## Window the pre-emphasized aa signal
windowed_aa = pre_emphasis_aa * hamming_window

## Find the FFT of the windowed aa signal
ft_aa = np.fft.fft(windowed_aa)
ft_aa = np.abs(ft_aa)
f_aa = np.fft.fftfreq(len(ft_aa), 1/sampling_rate)
```

2.1.1 Visualization of the narrowband magnitude spectrum slice

[ ]:
```
plt.figure(figsize=(30,5))
plt.subplot(1,2,1)
plt.plot(windowed_aa)
plt.xlabel('Time index')
plt.ylabel('Amplitude')
plt.title('Windowed audio signal (aa)')

plt.subplot(1,2,2)
plt.plot(f_aa, 20*np.log10(ft_aa))
plt.xlabel('Frequency')
plt.ylabel('DFT magnitude (dB)')
plt.title('DFT of pre-emphasized audio file (aa)')
```

[ ]:
```
Text(0.5, 1.0, 'DFT of pre-emphasized audio file (aa)')
```



2.1.2 Comments:

- The signal with windowed signal with pre-emphasis applied is shown on left and its Fourier magnitude spectrum is shown on right in the above figure.

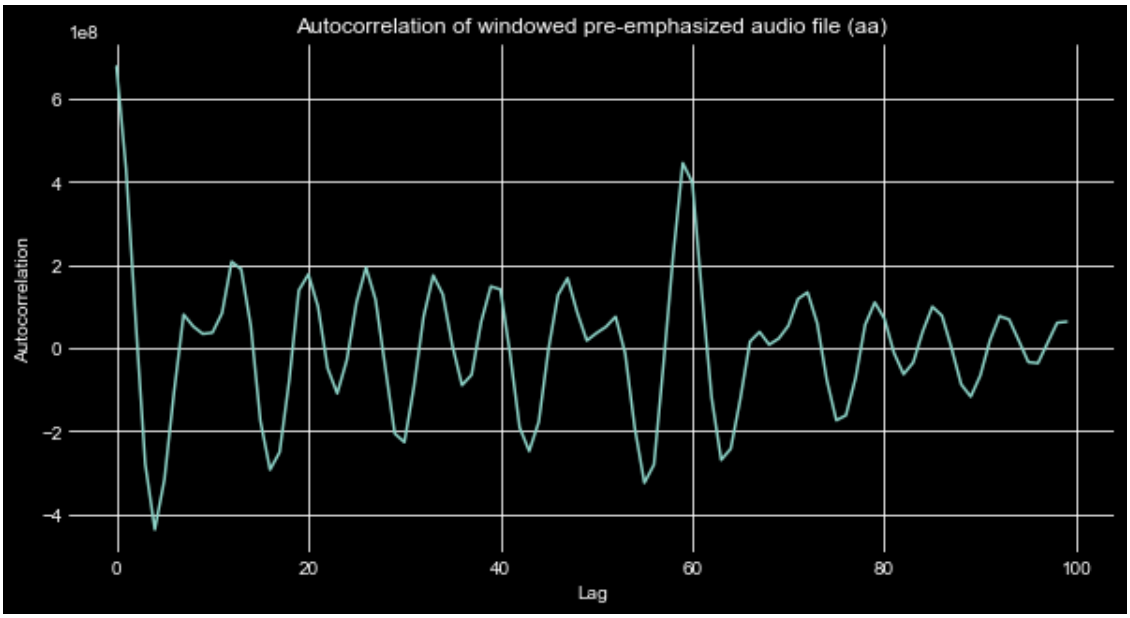
- The effect of pre-emphasis is shown clearly in the figure as we can see that the Fourrier magnitude spectrum does not fall off for the higher frequencies (which is the case for the original signal).

3 Part 3

3.1 With the same 30 ms segment of part 2, compute the autocorrelation coefficients required for LPC calculation at various p = 2,4,6,8,10. Use the Levinson-Durbin recursion to compute the LP coefficients from the autocorrelation coefficients. Plot error signal energy (i.e. square of gain) vs p.

[ ]:
```
## Let us compute the Auto-correlation of the windowed aa signal for lags of 0␣
↪to 100 index values
## However, we need only the first 10 coefficients for LPC calculation
def autocorrelation_func(signal, lags):
    """Function to compute the autocorrelation of a signal for a given range of␣
↪lags

    Args:
        signal (ndarray): The signal for which autocorrelation is to be computed
        lags (int): The range of lags for which autocorrelation is to be␣
↪computed

    Returns:
        ndarray: The autocorrelation of the signal for the given range of lags
    """
    autocorrelation_ = np.zeros(lags)
    for i in range(lags):
        autocorrelation_[i] = np.sum(signal[i:]*signal[:len(signal)-i])

    return autocorrelation_


autocorrelation = autocorrelation_func(windowed_aa, 100)
```

[ ]:
```
plt.figure(figsize=(10,5))
plt.plot(autocorrelation)
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.title('Autocorrelation of windowed pre-emphasized audio file (aa)')
plt.show()
```



3.1.1 COMMENTS:

- From the plot of the auto-correlation obtained above we can see that there are peaks at repeated lags of 60. These are the result of the harmonics present in the signal.

[ ]:
```
## Let us find the LP coefficients
## We will use the Levinson-Durbin algorithm
## We will use the first 10 coefficients


def Levinson_Durbin(autocorrelation, order):
    """Function to find the LP coefficients using the Levinson-Durbin algorithm
       Note: This function returns the LP coefficients such that the␣
 ↪coefficient at index 0 is 0 and should be ignored

    Args:
        autocorrelation (ndarray): Autocorrelation of the signal
        order (int): Order of the LP filter

    Returns:
        ndarray: LP coefficients such that the coefficient at index 0 is 0 and␣
 ↪should be ignored
    """
    ## Initialize the LPC coefficients
    lp_coefficients = np.zeros((order+1, order+1))

    ## Initialize the prediction error
```

```
    prediction_error = np.zeros(order+1)

    ## Initialize the reflection coefficients
    reflection_coefficients = np.zeros(order+1)

    prediction_error[0] = autocorrelation[0]

    # for i = 1
    reflection_coefficients[1] = autocorrelation[1]/prediction_error[0]
    lp_coefficients[1][1] = reflection_coefficients[1]

    prediction_error[1] = (1 -␣
 ↪reflection_coefficients[1]**2)*prediction_error[0]

    # for i = 2 to order

    for i in range(2, order+1):
        summation = 0
        for j in range(1, i):
            summation += lp_coefficients[i-1][j]*autocorrelation[i-j]

        reflection_coefficients[i] = (autocorrelation[i] - summation)/
 ↪prediction_error[i-1]

        lp_coefficients[i][i] = reflection_coefficients[i]

        for j in range(1, i):
            lp_coefficients[i][j] = lp_coefficients[i-1][j] -␣
 ↪reflection_coefficients[i]*lp_coefficients[i-1][i-j]

        prediction_error[i] = (1 -␣
 ↪reflection_coefficients[i]**2)*prediction_error[i-1]

    return lp_coefficients,prediction_error

lp_coefficients , error_energy = Levinson_Durbin(autocorrelation, 10)

gain = np.sqrt(error_energy)                # Gain of the LP filter which is␣
 ↪the square root of the prediction error
```

[ ]:
```
orders = [2,4,6,8,10]

for order in orders:
    print("The coefficients for p="+str(order)+" are:",␣
 ↪str(lp_coefficients[order][1:order+1]))
```

```
The coefficients for p=2 are: [ 0.96454776 -0.51474629]
```

```
The coefficients for p=4 are: [ 0.68322148 -0.13507646 -0.28905557 -0.16557719]
The coefficients for p=6 are: [ 0.73630801 -0.14236591 -0.31958052 -0.28524925
0.28348265 -0.18881888]
The coefficients for p=8 are: [ 0.5881573  -0.11510746 -0.22581153 -0.54813899
0.03861646 -0.09802675
  0.21389618 -0.58065459]
The coefficients for p=10 are: [ 0.71103761 -0.06174574 -0.2354676  -0.55416698
0.16613212  0.05255368
  0.2772599  -0.72267503  0.17179372  0.15318719]
```

3.1.2 Plot of the Error Energy Function

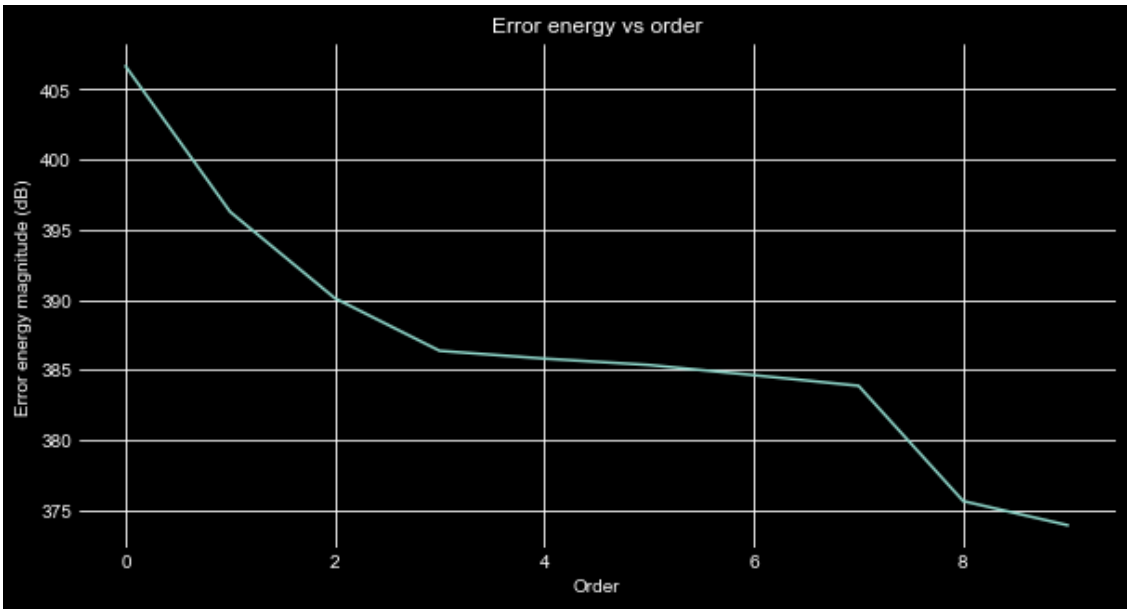
The error energy function is the sum of the squared prediction errors for each lag. Although I have calculated the error energy in the implementation of the Levinson-Durbin recursion, I will calculate it again here from the first principles i.e. using the equation

$$Error\ energy = r[0] - \sum_k a_k \times r[k]$$

where $r[k]$ is the autocorrelation coefficient at lag k and a_k is the coefficient of the kth order LPC filter.

[ ]:
```
error_energy = np.zeros(10)
error_energy[0] = autocorrelation[0]

for i in range(1,10):
    error_energy[i] = autocorrelation[0] - np.dot(lp_coefficients[i][1:i+1].
 ↪reshape(-1),autocorrelation[1:i+1])

plt.figure(figsize=(10,5))
plt.plot(20*np.log(error_energy))
plt.xlabel('Order')
plt.ylabel('Error energy magnitude (dB)')
plt.title('Error energy vs order')
```

[ ]:
```
Text(0.5, 1.0, 'Error energy vs order')
```



3.1.3 Comment:

We can easily see from the plot that the error energy function is decreasing with increasing order of the LPC filter.

4 Part 4

4.1 Show the pole-zero plots of the estimated all-pole filter for p=6,10; comment

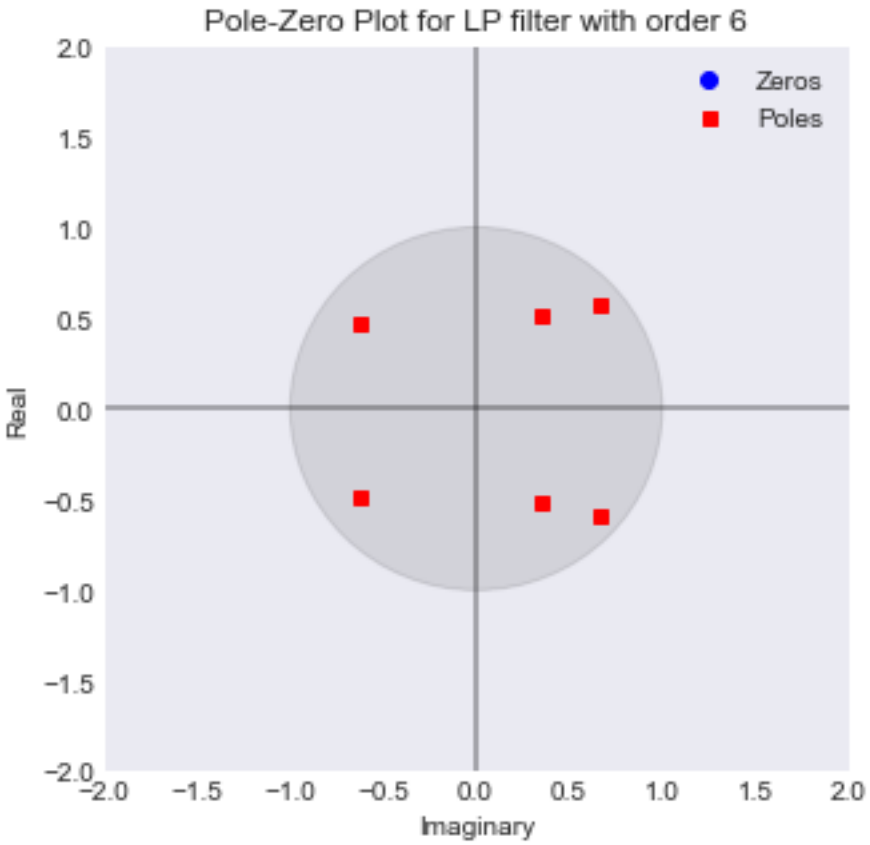
[ ]:
```
from scipy import signal
plt.style.use('seaborn-darkgrid')
def poleZeroPlot(b,a,order):
    """Function to plot the pole-zero plot of the LP filter

    Args:
        b (float): The numerator coefficients of the LP filter
        a (ndarray): The coefficients of the LP filter
        order (int): The order of the LP filter
    """
    poles = np.zeros(len(a))
    poles[0]=1
    poles[1:len(a)] = -a[1:len(a)]
    b=[b]
    z, p, k = signal.tf2zpk(b, poles)
    p = p[p!=0]

    fig = plt.figure(figsize=(5,5))
```

```
    ax=fig.add_subplot(1, 1, 1)

    plt.plot(np.real(z), np.imag(z), 'ob')
    plt.plot(np.real(p), np.imag(p), 'sr',markersize=5,fillstyle="full")
    circ = plt.Circle((0, 0), radius=1,facecolor='None',color='black',␣
↪ls='solid', alpha=0.1)
    ax.add_patch(circ)
    plt.axhline(0,color='black',alpha=0.4)
    plt.axvline(0,color='black',alpha=0.4)
    plt.ylim((-2.0, 2.0))
    plt.xlim((-2.0,2.0))
    plt.legend(['Zeros', 'Poles'])
    plt.ylabel('Real')
    plt.xlabel('Imaginary')
    plt.title('Pole-Zero Plot for LP filter with order '+str(order))
    plt.grid()

    plt.show()
```

4.1.1 For order = 6

[ ]:
```
poleZeroPlot(gain[6],lp_coefficients[6][:],6)
```

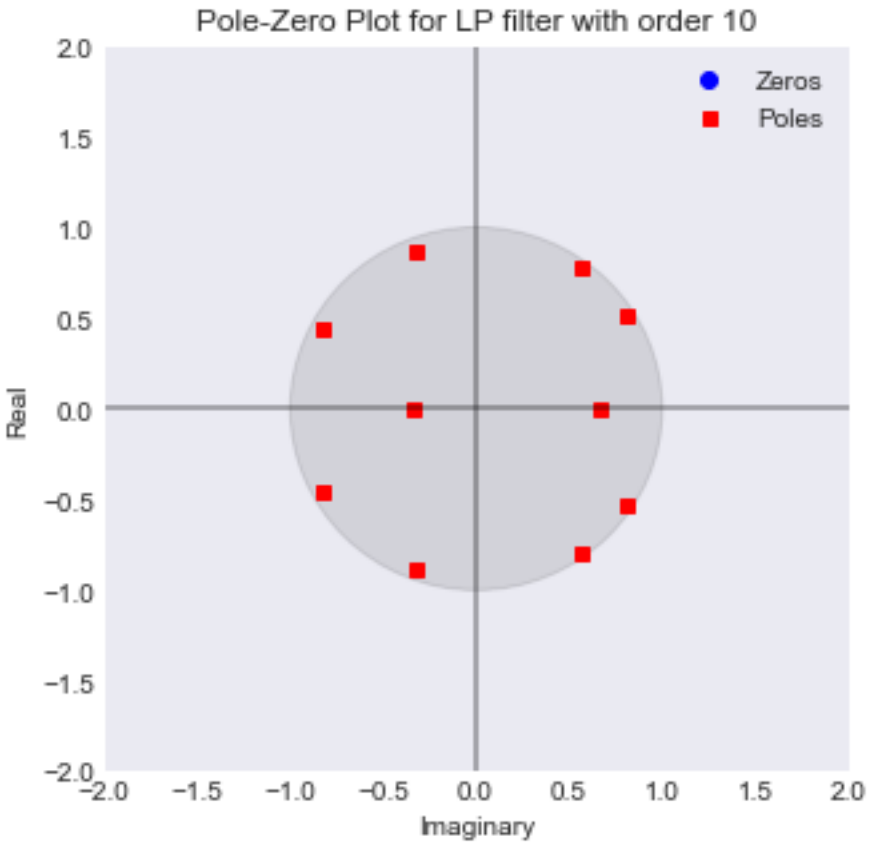
```
C:\Users\Vinit\AppData\Local\Temp/ipykernel_4544/3354072316.py:16: UserWarning:
Setting the 'color' property will override the edgecolor or facecolor
properties.
  circ = plt.Circle((0, 0), radius=1,facecolor='None',color='black', ls='solid',
alpha=0.1)
```



4.1.2 For order = 10

[ ]:
```
poleZeroPlot(gain[10],lp_coefficients[10][:],10)
```

```
C:\Users\Vinit\AppData\Local\Temp/ipykernel_4544/3354072316.py:16: UserWarning:
Setting the 'color' property will override the edgecolor or facecolor
properties.
  circ = plt.Circle((0, 0), radius=1,facecolor='None',color='black', ls='solid',
alpha=0.1)
```



4.2 Comments:

- Firstly, all the poles in both the cases are inside the unit circle. This implies that the LP filter design is stable.
- By looking at the values and also by the graph it is very evident that the poles for p=10 are closer to the unit circle as compared to the poles of the p=6.
- This is due to the fact that as we increase p, the actual specturm is closely traced. Hence, as spectrum of larger p has lesser bandwidth, they are closer to the unit circle.
- Also, there are two poles for p=10 which are real. For the case of p=6, there are no real poles.

[ ]:
```
plt.style.use('dark_background')
```

5 Part 5

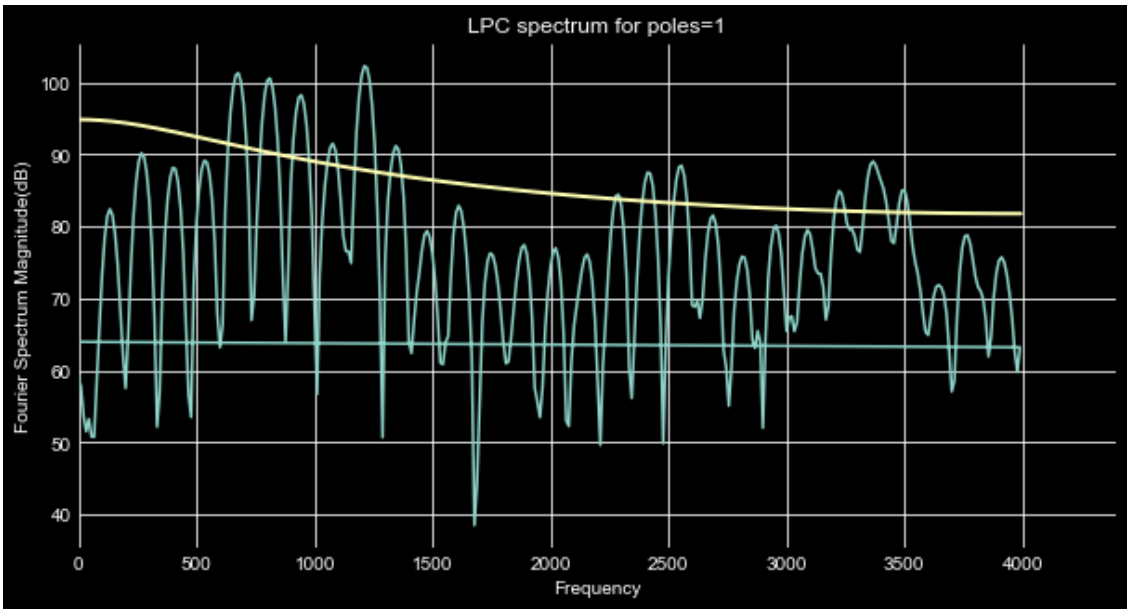
5.1 Compute the gain and plot the LPC spectrum magnitude (i.e. the dB magnitude frequency response of the estimated all-pole filter) for each order "p". Comment on the characteristics of the spectral envelope estimates. Comment on their shapes with reference to the short-time magnitude spectrum computed in part 2

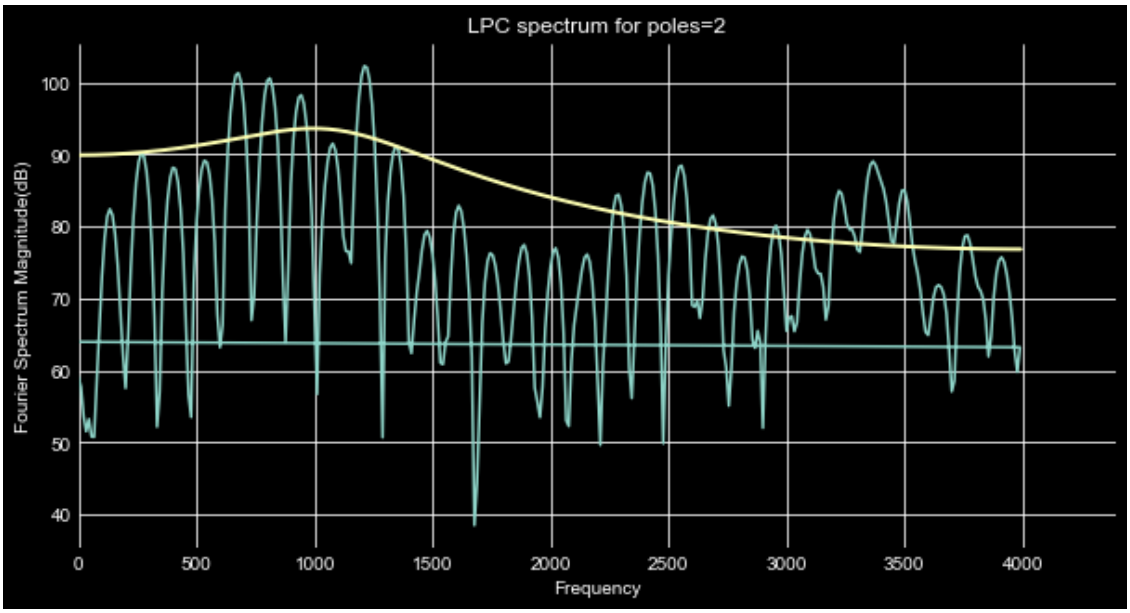
[ ]:
```
gain                # We have already computed the gain for the LP filter
```

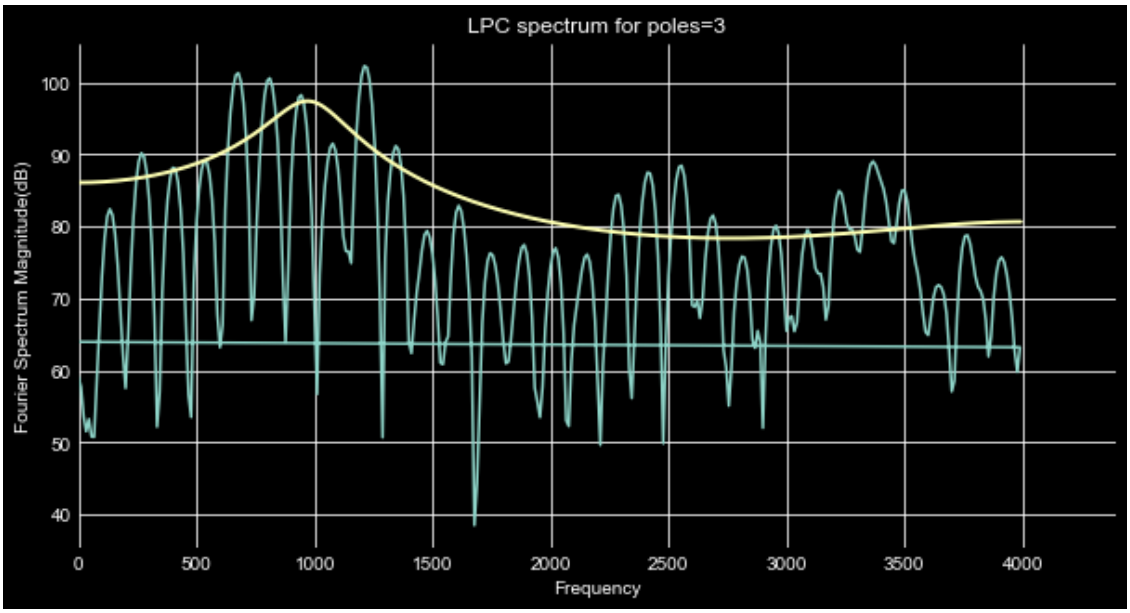
[ ]:
```
array([26014.46894783, 20058.51601538, 17197.01933469, 15657.81321431,
       15441.68580839, 15267.4599599 , 14992.82783309, 14712.31310423,
       11978.02988378, 11472.46446395, 11337.05712869])
```

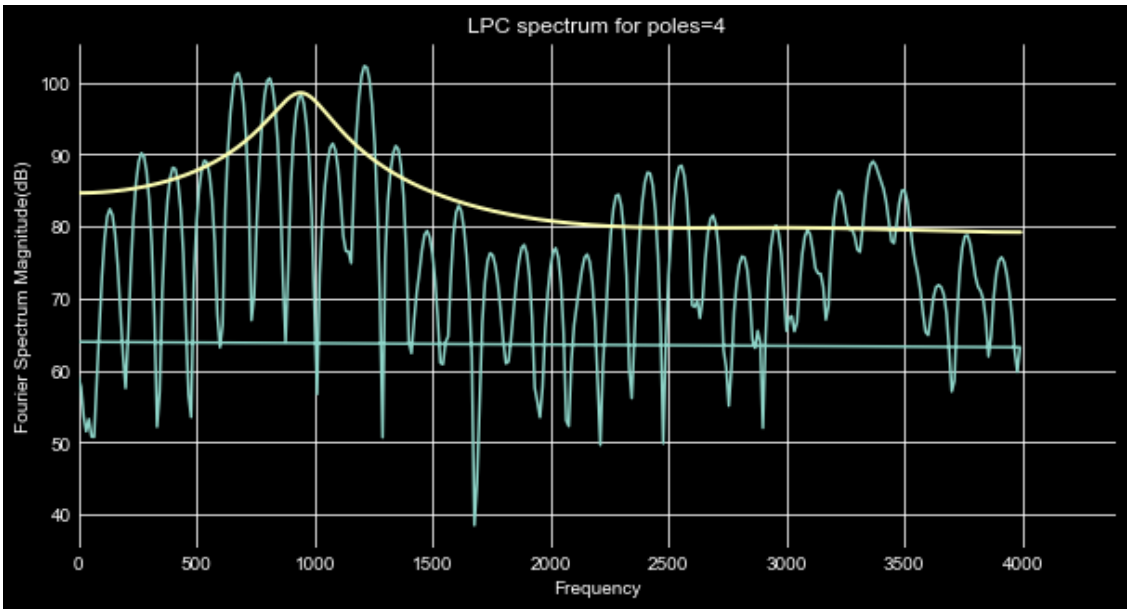
[ ]:
```
from scipy.signal import freqz

def plot_spectrum(order,lp_coefficients_,gain_):
    for i in range(1,order+1):
        poles = np.zeros_like(lp_coefficients_[i])
        poles[0] = 1
        poles[1:] = -lp_coefficients_[i][1:]
        w,h = freqz(gain_[i],poles)

        plt.figure(figsize=(10,5))
        plt.title("LPC spectrum for poles="+str(i))
        plt.plot(f_aa, 20*np.log10(np.abs(ft_aa)), label = 'Original Spectrum')
        plt.plot(w*sampling_rate/(2*np.pi),20*np.log10(abs(h)),linewidth=2,␣
 ↪label = "LPC Estimate Spectrum")
        plt.ylabel("Fourier Spectrum Magnitude(dB)")
        plt.xlabel("Frequency")
        plt.xlim(xmin=0)
        plt.show()
```

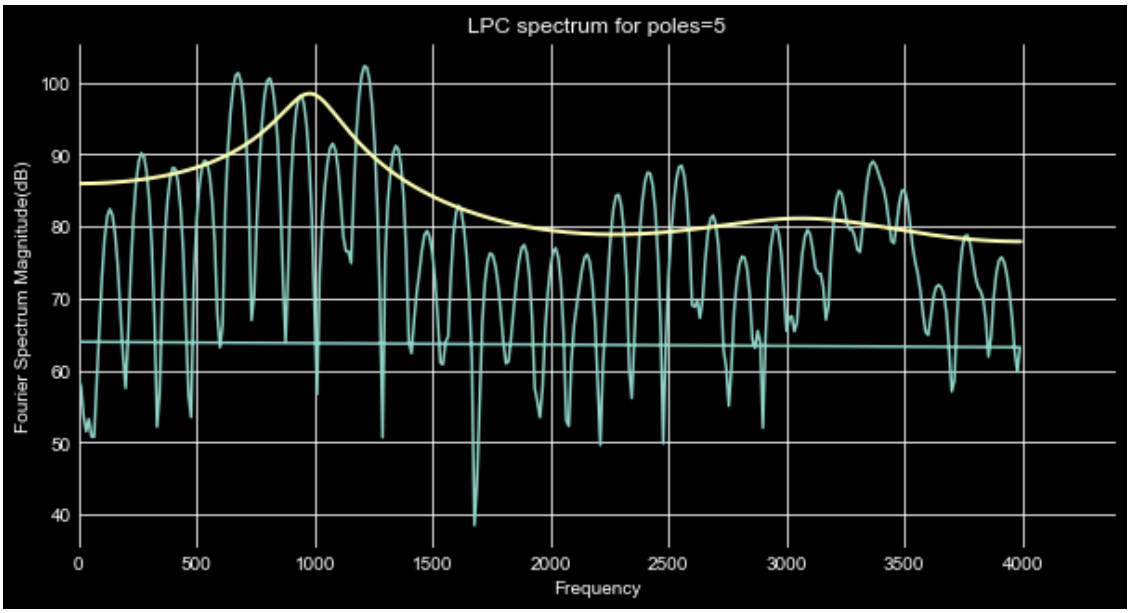
[ ]:
```
order = 10

plot_spectrum(order,lp_coefficients,gain)
```

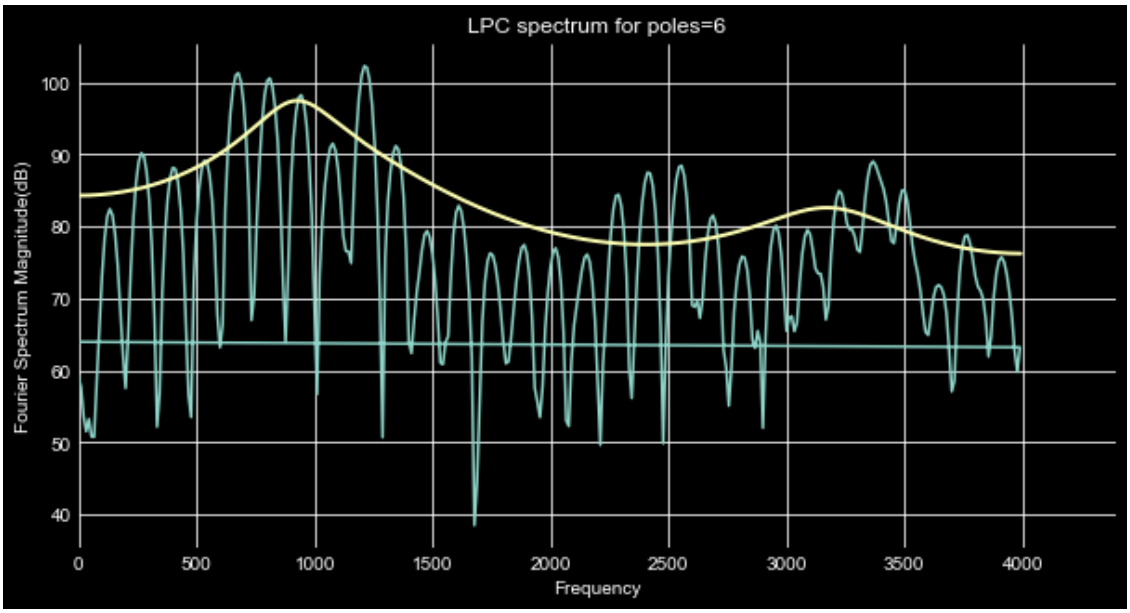

LPC spectrum for poles=1
Fourier Spectrum Magnitude(dB)
100
90
80
70
60
50
40
0
500
1000
1500
2000
2500
3000
3500
4000
Frequency

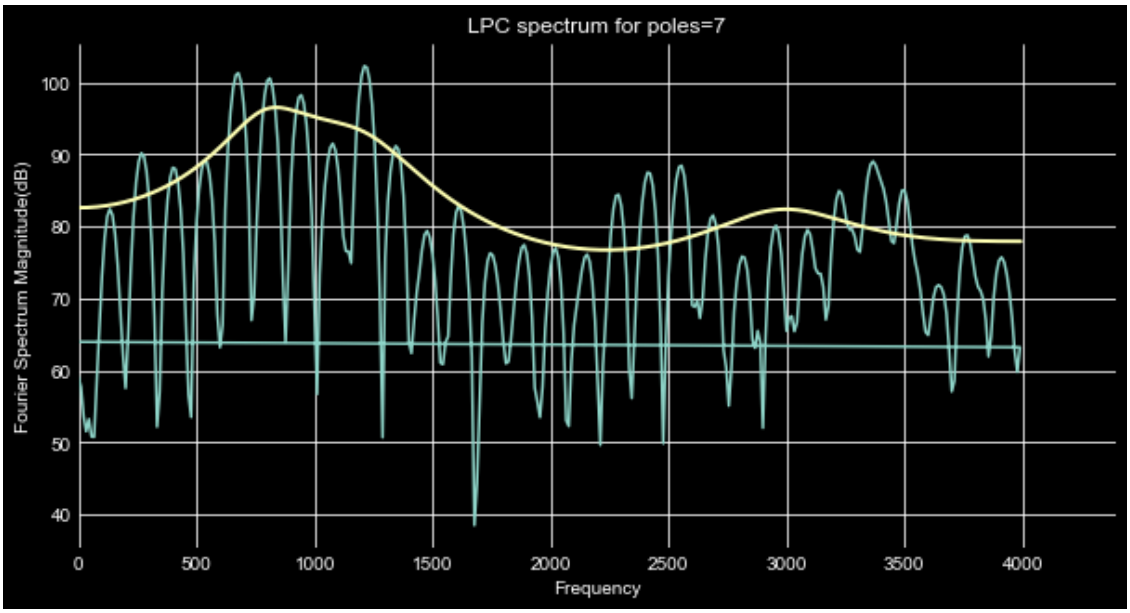

LPC spectrum for poles=2
Fourier Spectrum Magnitude(dB)
100
90
80
70
60
50
40
0
500
1000
1500
2000
2500
3000
3500
4000
Frequency

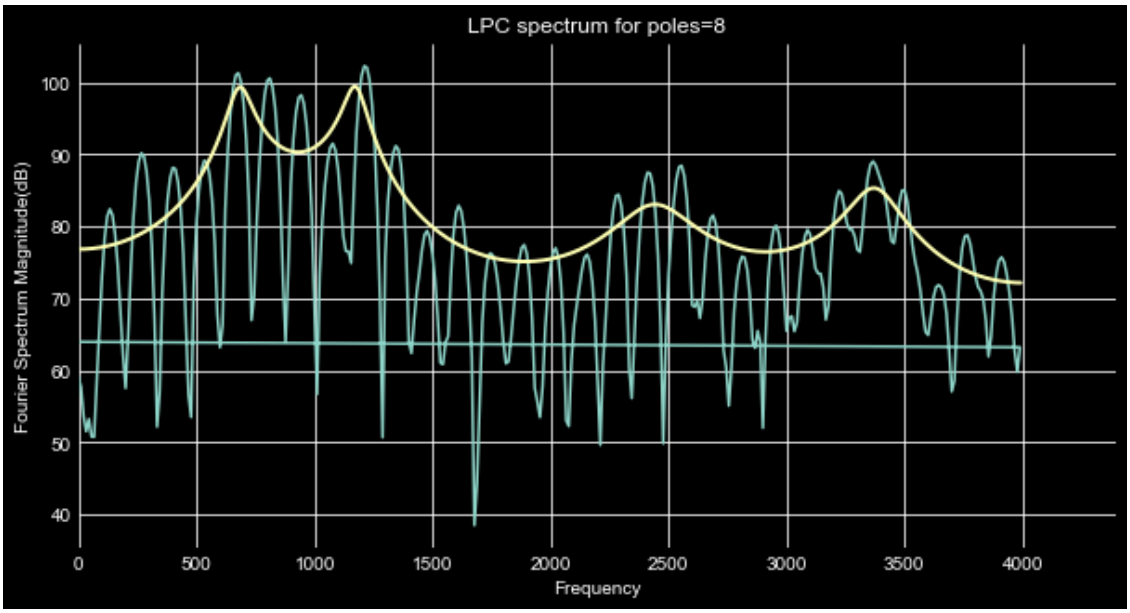

LPC spectrum for poles=3
Fourier Spectrum Magnitude(dB)
100
90
80
70
60
50
40
0
500
1000
1500
2000
2500
3000
3500
4000
Frequency

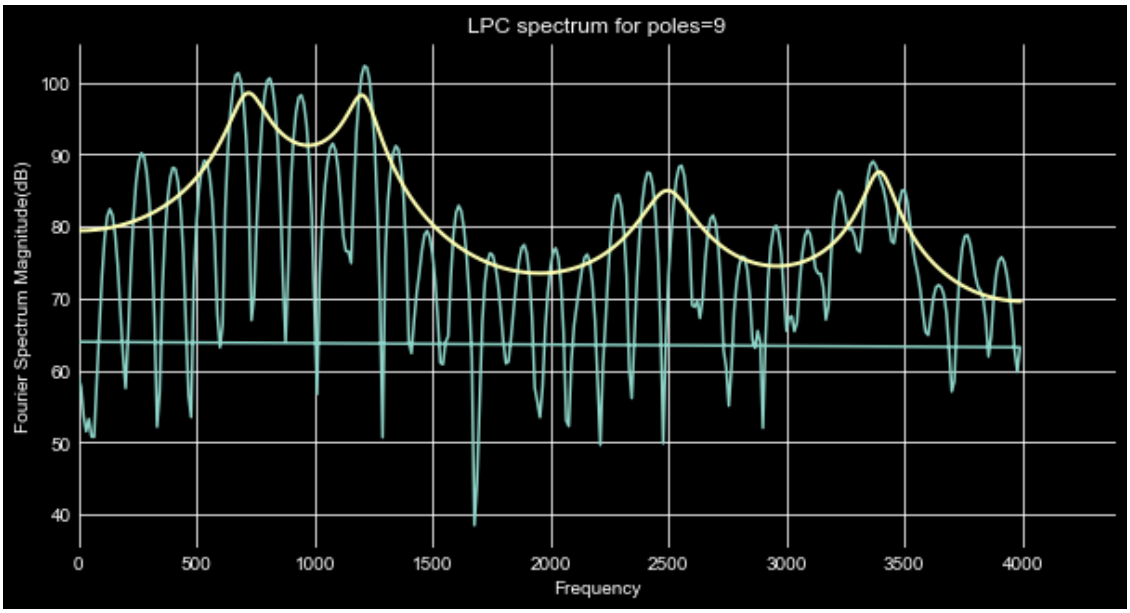

LPC spectrum for poles=4
Fourier Spectrum Magnitude(dB)
100
90
80
70
60
50
40
0
500
1000
1500
2000
2500
3000
3500
4000
Frequency

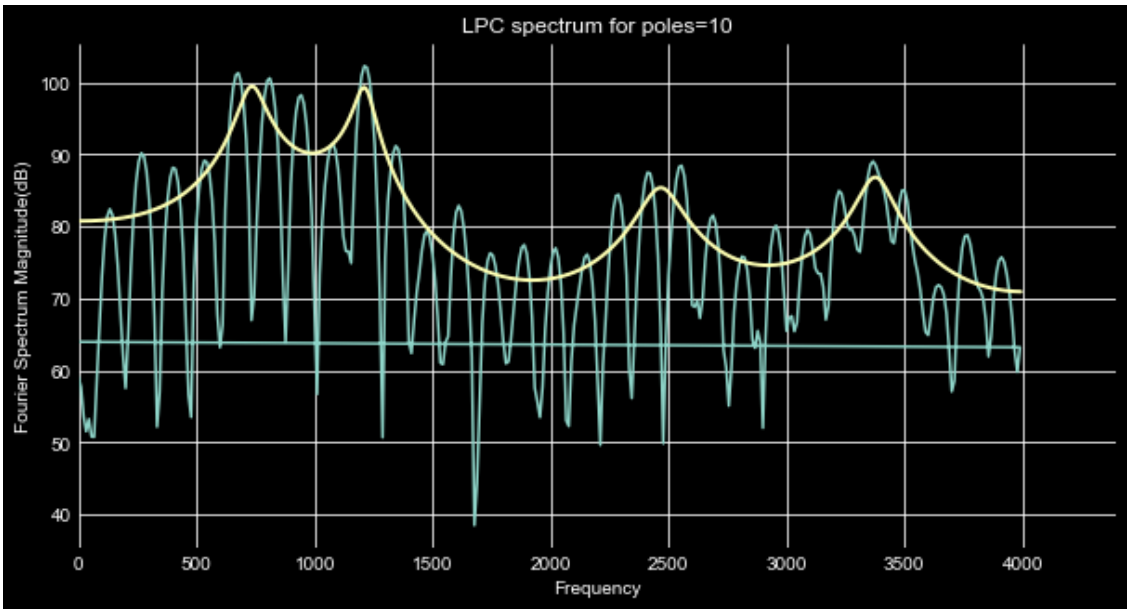

LPC spectrum for poles=5
Fourier Spectrum Magnitude(dB)
100
90
80
70
60
50
40
0
500
1000
1500
2000
2500
3000
3500
4000
Frequency


LPC spectrum for poles=6
Fourier Spectrum Magnitude(dB)
100
90
80
70
60
50
40
0
500
1000
1500
2000
2500
3000
3500
4000
Frequency


LPC spectrum for poles=7
Fourier Spectrum Magnitude(dB)
100
90
80
70
60
50
40
0
500
1000
1500
2000
2500
3000
3500
4000
Frequency


LPC spectrum for poles=8
Fourier Spectrum Magnitude(dB)
100
90
80
70
60
50
40
0
500
1000
1500
2000
2500
3000
3500
4000
Frequency





5.1.1 Commments:

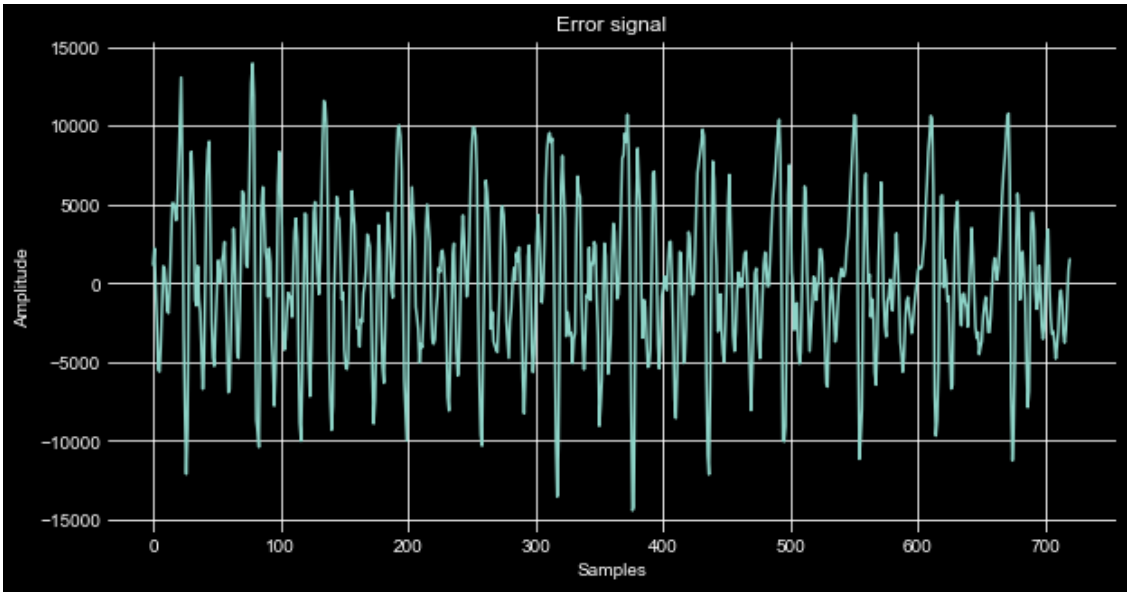
- We can clearly see from the above plots that the estimated spectrum closely matches the short-time magnitude spectrum.
- Further as the order increases, the estimated spectrum closely matches the short-time magnitude spectrum.
- For order 8, the estimated spectrum captures all the formants present in the signal.

- Also, the fact mentioned earlier that the bandwith of the spectrum is less (at all the pole locations) for higher order filters is also evident from the above plots.
- Also, the estimated spectrum for order 10 does not show the presence of poles which are real.

6 Part 6

6.1 Based on the 10th-order LP coefficients, carry out the inverse filtering of the /a/ vowel segment to obtain the residual error signal. Can you measure the pitch period of the voiced sound from the residual waveform? Use the acf to detect the pitch. Compare the acf plots of the original speech and residual signals.

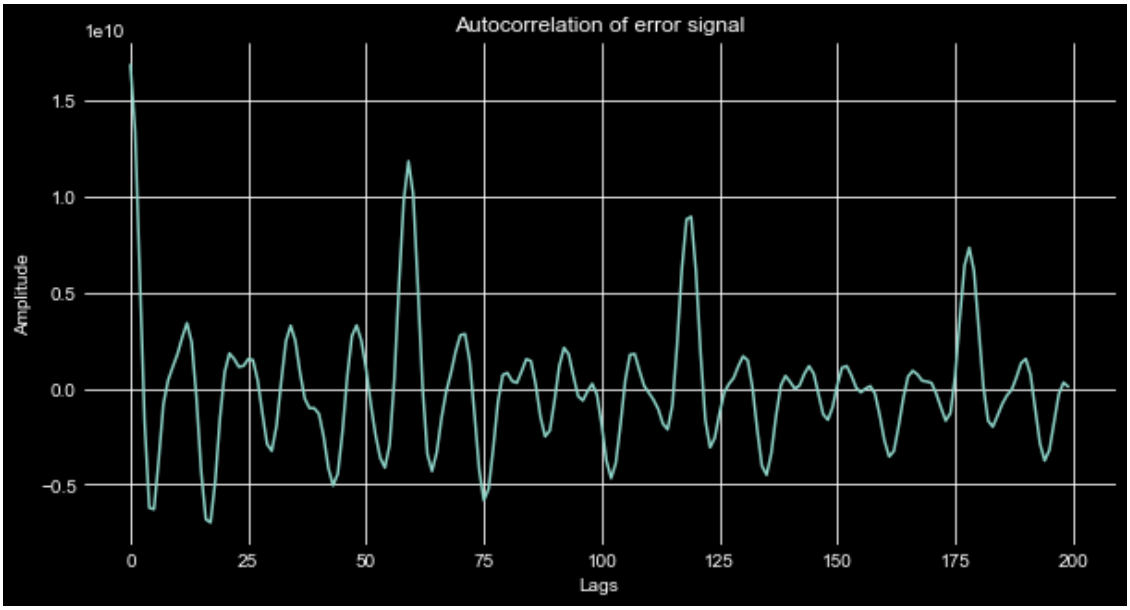
[ ]:
```
lp_10 = lp_coefficients[10][1:]

# Inverse filtering using the LPC coefficients for windowed_aa signal

estimated_aa = np.zeros(len(windowed_aa))

for i in range(len(estimated_aa)):
    summation = 0
    for j in range(0,order):
        # ignore the negative indices
        if i-j >= 0:
            summation += lp_10[j]*windowed_aa[i-j]

    estimated_aa[i] = summation

error_signal = aa - estimated_aa


plt.figure(figsize=(10,5))
plt.plot(error_signal)
plt.xlabel('Samples')
plt.ylabel('Amplitude')
plt.title('Error signal')
plt.show()
```



6.1.1 COMMENT:

- We can see from the above plot that the error signal has larger magnitude at the starting of the window. Also the error will be higher near the glottal pulses (harmonics).

[ ]:
```
## Find the autocorrelation of the error signal to get the pitch period

autocorrelation_error = autocorrelation_func(error_signal,200)

plt.figure(figsize=(10,5))
plt.plot(autocorrelation_error)
plt.xlabel('Lags')
plt.ylabel('Amplitude')
plt.title('Autocorrelation of error signal')
plt.show()
```



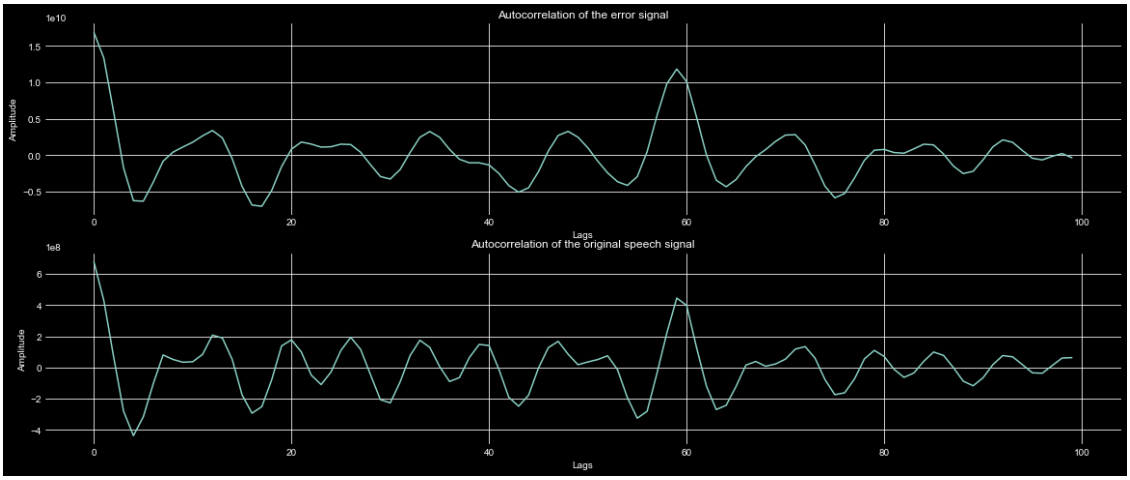
From the figure we can see that the autocorrelation function has peaks after a periodic lag. Visually, we can see that the lag is roughly 60 samples. Hence, as we know that the sampling rate is 8 kHz, the pitch period can be found out as follows.

$$Pitch = Sampling\,rate\,/\,60$$

$$Pitch = 8000/60 = 133.33Hz$$

6.1.2 Now we compare the autocorrelation of the residual error signal with the original speech signal.

[ ]:
```
plt.figure(figsize=(20,8))
plt.subplot(2,1,1)
plt.plot(autocorrelation_error[:100])
plt.xlabel('Lags')
plt.ylabel('Amplitude')
plt.title('Autocorrelation of the error signal')

plt.subplot(2,1,2)
plt.plot(autocorrelation)
plt.xlabel('Lags')
plt.ylabel('Amplitude')
plt.title('Autocorrelation of the original speech signal')
plt.show()
```



6.1.3 Comments:

- From the above plots we can see that the auto-correlation function for the error signal and that of the original speech signal have spikes at period lags of 60. These spikes correspond to the harmonics present in the original speech signal.
- The other peaks which are present in the auto-correlation function of the residual error signal are also present in the auto-correlation function of the original speech signal however, the relative amplitude of these peaks is slightly less.

7 Part 7

7.1 LP re-synthesis: We analysed a natural speech sound /a/ above. Using a suitable set of parameter estimates as obtained there, we wish to reconstruct the sound.

7.2 That is, use the best estimated LP filter with an ideal impulse train of the estimated pitch period as source excitation. Carry out de-emphasis on the output waveform. Set the duration of the synthesized sound to be 300 ms at 8 kHz sampling frequency and view the waveform as well as listen to your created sound.

7.3 Comment on the similarity with the original sound. Try out voice modification using this analysis-synthesis method (e.g. change the voice pitch).

7.4 Solution

The best estimated LP filter is the one with order 10. Hence, we can use the LP coefficients obtained from the above analysis to synthesize the speech. Also, the estimated pitch period is 133.33 Hz. Hence, we can use the impulse train of 133.33 Hz or 60 samples (keeping the sampling rate to be same) as the excitation signal.

```
[ ]: def resynthesize(duration,pitch, gain, lp_coefficients, sampling_rate):
        """Function to resynthesize the speech signal

        Args:
            duration (float): The duration of the speech signal in seconds
            pitch (int): The pitch of the speech signal
            gain (float): The gain of the speech signal
            lp_coefficients (ndarray): The LP coefficients
            sampling_rate (int): The sampling rate of the speech signal

        Returns:
            impulse (ndarray): The impulse train used as excitation signal
            synthesized_speech (ndarray): The synthesized speech signal
            timeSample (ndarray): The time samples of the synthesized speech signal
        """
        impulse = np.zeros(int(duration*sampling_rate))
        pitch_samples = np.floor(sampling_rate/pitch)
        for i in range(len(impulse)):
            if (i%pitch_samples == 0):
                impulse[i] = 1

        lp_coefficients = -lp_coefficients
        lp_coefficients = lp_coefficients.tolist()
        den_terms = [1] + lp_coefficients

        synthesized_speech = signal.lfilter([gain],den_terms,impulse)    # Using␣
    ↪inbuilt function for filtering since this has already been implemented as a␣
    ↪part of computing_assg 1
        synthesized_speech = np.int16(synthesized_speech/np.max(np.
    ↪abs(synthesized_speech)) * 32767)
        timeSample = np.linspace(0, duration, int(duration*sampling_rate))

        return impulse, synthesized_speech, timeSample
```

7.5 For de-emphasis, we can use the inverse z transform.

Basically, we want to filter the output with a filter of the form

$$H(z) = \frac{1}{1 - \alpha z^{-1}}$$

where α is the pre-emphasis coefficient. For this assignment, we will use $\alpha = 0.95$.

Hence, in time domain, we have

$$\frac{Y(z)}{X(z)} = \frac{1}{1 - \alpha z^{-1}}$$

where $Y(z)$ is the output signal and $X(z)$ is the excitation signal z- transform.

$$\implies Y(z) \times (1 - \alpha z^{-1}) = X(z)$$

Taking inverse z transform, we get

$$y[n] - \alpha y[n-1] = x[n]$$

where $y[n]$ is the output signal and $x[n]$ is the excitation signal.

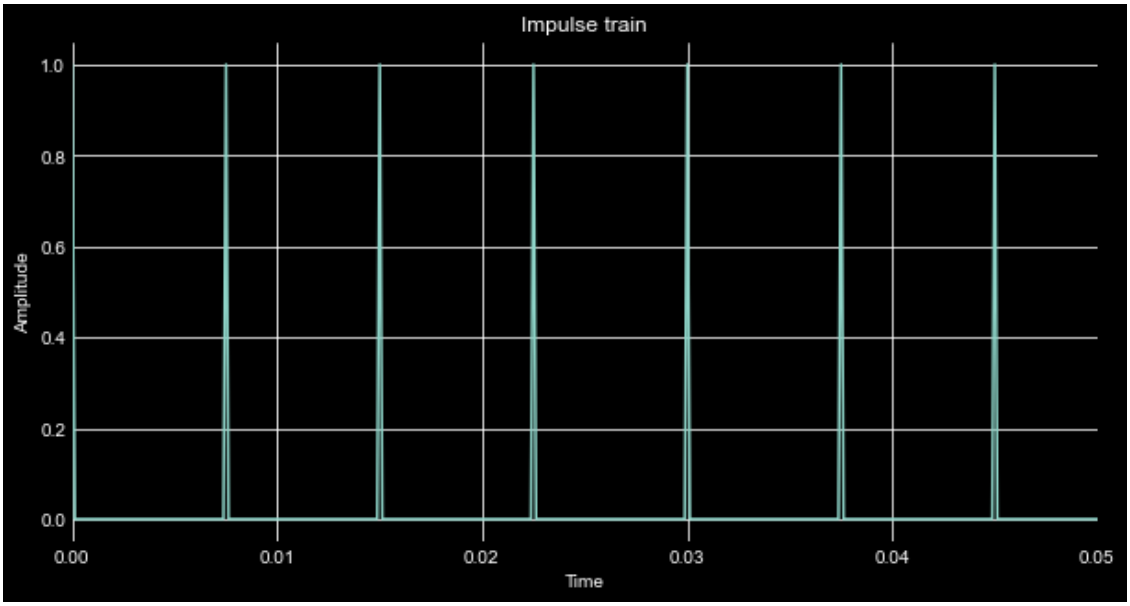
$$\implies y[n] = x[n] + \alpha y[n-1]$$

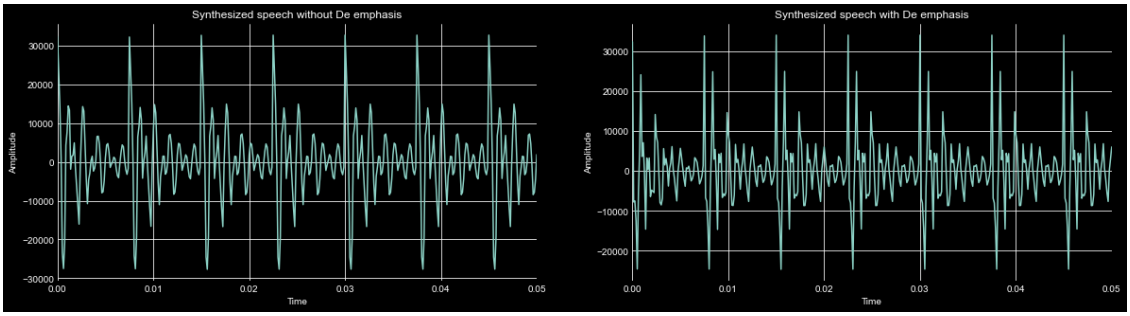
[ ]:
```
def de_emphasis(signal,alpha):
    """Function to apply de-emphasis to the speech signal

    Args:
        signal (ndarray): The speech signal
        alpha (float): The de-emphasis factor

    Returns:
        ndarray: The de-emphasized speech signal
    """
    output = np.zeros(len(signal))
    output[0] = signal[0] # Assuming initial rest
    for i in range(1,len(signal)):
        output[i] = signal[i] - alpha*signal[i-1]

    return output
```

7.5.1 Now we can synthesize the speech.

[ ]:
```
impulse_train, synthesized_aa, time = resynthesize(0.5,133,gain[10],lp_10,8000)

de_emphasis_aa = de_emphasis(synthesized_aa,0.95)

# Let us write the synthesized speech to a file
import scipy.io.wavfile as wav

wav.write('synthesized_speech_133.wav',8000,np.int16(de_emphasis_aa))

# Now, let us look at the waveform of the synthesized speech
plt.figure(figsize=(10,5))
plt.plot(time, impulse_train)
plt.xlim(0,0.05)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('Impulse train')

plt.figure(figsize=(20,5))
```

```
plt.subplot(1,2,1)
plt.xlim(0,0.05)
plt.plot(time, synthesized_aa)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('Synthesized speech without De emphasis')

plt.subplot(1,2,2)
plt.plot(time, de_emphasis_aa)
plt.xlim(0,0.05)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('Synthesized speech with De emphasis')
plt.show()
```





```
[ ]: ### Button press for the synthesized speech
     IPython.display.Audio("synthesized_speech_133.wav")
```

```
[ ]: <IPython.lib.display.Audio object>
```

7.5.2 COMMENT:

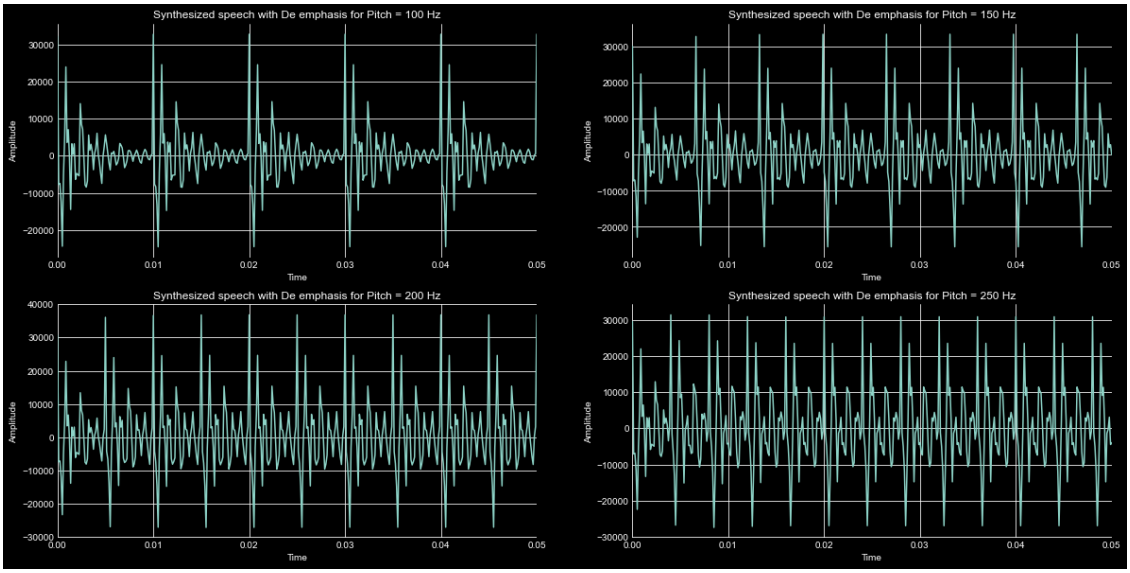
- We can clearly see that the synthesized speech is very similar to the original speech.

7.5.3 Now, let us see the effect of changing pitch

We will try the following pitch changes: - 100 Hz - 150 Hz - 200 Hz - 250 Hz

```
[ ]: # Pitch = 100 Hz
     impulse_train, synthesized_aa_100, time_100 = resynthesize(0.
      ↪5,100,gain[10],lp_10,8000)
     de_emphasis_aa_100 = de_emphasis(synthesized_aa_100,0.95)
     wav.write('synthesized_speech_100.wav',8000,np.int16(de_emphasis_aa_100))

     # Pitch = 150 Hz
     impulse_train, synthesized_aa_150, time_150 = resynthesize(0.
      ↪5,150,gain[10],lp_10,8000)
     de_emphasis_aa_150 = de_emphasis(synthesized_aa_150,0.95)
     wav.write('synthesized_speech_150.wav',8000,np.int16(de_emphasis_aa_150))

     # Pitch = 200 Hz
     impulse_train, synthesized_aa_200, time_200 = resynthesize(0.
      ↪5,200,gain[10],lp_10,8000)
     de_emphasis_aa_200 = de_emphasis(synthesized_aa_200,0.95)
     wav.write('synthesized_speech_200.wav',8000,np.int16(de_emphasis_aa_200))

     # Pitch = 250 Hz
     impulse_train, synthesized_aa_250, time_250 = resynthesize(0.
      ↪5,250,gain[10],lp_10,8000)
     de_emphasis_aa_250 = de_emphasis(synthesized_aa_250,0.95)
     wav.write('synthesized_speech_250.wav',8000,np.int16(de_emphasis_aa_250))

     # Now, let us look at the waveform of the synthesized speech

     plt.figure(figsize=(20,10))
     plt.subplot(2,2,1)
     plt.plot(time_100, de_emphasis_aa_100)
     plt.xlim(0,0.05)
     plt.xlabel('Time')
     plt.ylabel('Amplitude')
     plt.title('Synthesized speech with De emphasis for Pitch = 100 Hz')

     plt.subplot(2,2,2)
```

```
plt.plot(time_150, de_emphasis_aa_150)
plt.xlim(0,0.05)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('Synthesized speech with De emphasis for Pitch = 150 Hz')

plt.subplot(2,2,3)
plt.plot(time_200, de_emphasis_aa_200)
plt.xlim(0,0.05)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('Synthesized speech with De emphasis for Pitch = 200 Hz')

plt.subplot(2,2,4)
plt.plot(time_250, de_emphasis_aa_250)
plt.xlim(0,0.05)
plt.xlabel('Time')
plt.ylabel('Amplitude')
plt.title('Synthesized speech with De emphasis for Pitch = 250 Hz')
plt.show()
```



7.5.4 Observe the effect of changing pitch on the synthesized speech.

Pitch = 100 Hz

[ ]:
```
IPython.display.Audio("synthesized_speech_100.wav")
```

[ ]:
```
<IPython.lib.display.Audio object>
```

Pitch = 150 Hz

```
[ ]: IPython.display.Audio("synthesized_speech_150.wav")
```

```
[ ]: <IPython.lib.display.Audio object>
```

Pitch = 200 Hz

```
[ ]: IPython.display.Audio("synthesized_speech_200.wav")
```

```
[ ]: <IPython.lib.display.Audio object>
```

Pitch = 250 Hz

```
[ ]: IPython.display.Audio("synthesized_speech_250.wav")
```

```
[ ]: <IPython.lib.display.Audio object>
```

7.5.5 COMMNENT:

- The effect of changing the pitch is clear from the sounds.