Speech Signal Processing(EE679) Assignment 2

Akshay Bajpai(193079002)

31st October 2020

1 Report Structure

For each of the questions , I have written the basic algorithm followed to implement the solution , followed by the code and the output graphs. The codes have been written in octave and the corresponding output files(**plots and .wav files**) are in the outputs folder submitted .

1.1 Question 1

Given the speech segment (aa.wav) extracted from the word "pani" in "machali.wav" (male voice), sampled at 8 kHz, do the following.

1. Apply pre-emphasis to the signal.

Solution :

Algorithm :

1. Pre emphasis is applying the filter with 1 zero with coefficient α (0.95 - 0.99) chosen as 0.98 here.

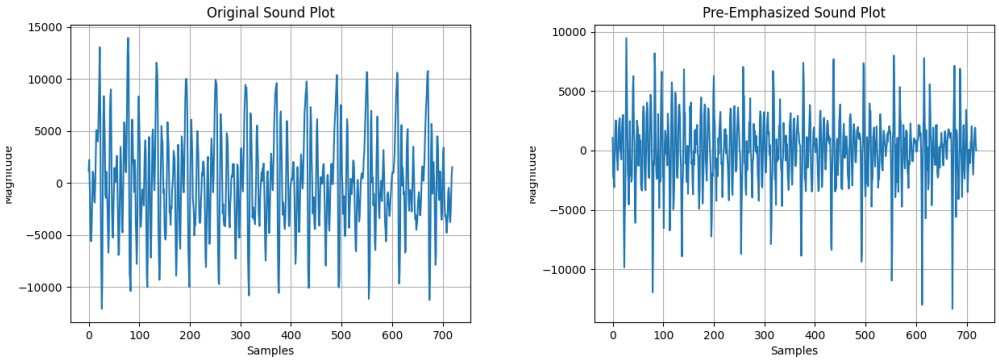
$$H(z) = 1 - \alpha z^{-1} \tag{1}$$

2. The above equation is applied in the code using difference equation using the equation below .

$$y[n] = x[n] - \alpha x[n-1] \tag{2}$$

Code :
Pre-Emphasis Function :

```
def pre_emphasize(sound,alpha):

    pre_sound = np.zeros(len(sound))
    for i in range(1,len(sound)):
        pre_sound[i-1] = sound[i] - alpha*sound[i-1]
    return pre_sound
```



(a) (b)

Figure 1: (a)Original Input Waveform (b)Pre-Emphasized Input Waveform

1.2 Question 2

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.

Solution :

Algorithm :

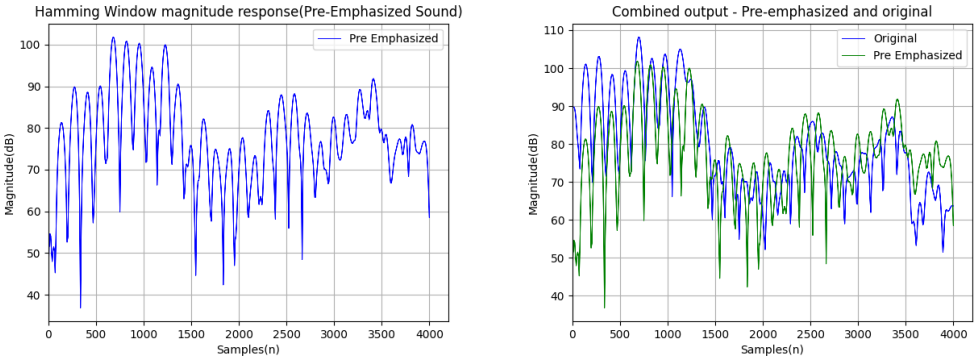
1. Create a hamming window length of 30ms
2. Calculate the center of the input waveform and apply hamming window around that sample.
3. Calculate 1024 point DFT of the windowed segment and plot.

Code :
Hamming Window :

```
#This is a section of the code from the main script

win_length = 30
window_size = int(win_length*fs/1000)
center = int(len(pre_sound)/2)
window_signal_pre = pre_sound[center - window_size : center] * np.hamming(window_size
    )
window_signal = sound[:window_size] * np.hamming(window_size)

dft_orig = [fft(window_signal, n=1024)]
dft_pre = [fft(window_signal_pre, n=1024)]
freq = [fftfreq(dft_orig[0].shape[-1], 1/fs)]

combined = [*dft_orig,*dft_pre]
freq = [*freq,*freq]
plot_spectrum(dft_orig,freq,["Original"],title_="Hamming Window magnitude response(
    Original Sound)",plots=1)
plot_spectrum(dft_pre,freq,["Pre Emphasized"],title_="Hamming Window magnitude
    response(Pre-Emphasized Sound)",plots=1)
plot_spectrum(combined,freq,["Original","Pre Emphasized"],title_="Combined output -
    Pre-emphasized and original",plots=2)
```

Plotting the spectrum :

```
def plot_spectrum(dft_,freq,names,title_,plots):
    colors = ['b','g','k','r']
    plt.figure()
    for i in range(plots):
        len_ = int(len(dft_[i])/2)
        plt.plot(abs(freq[i]),20*log10(abs(dft_[i])),colors[i],linewidth=0.7)

    plt.legend([i for i in names])
    plt.xlim(xmin=0)
    plt.grid("True")
    plt.ylabel("Magnitude(dB)")
    plt.xlabel("Samples(n)")
    plt.title(title_)
    plt.savefig('outputs/'+str(plots)+title_+".png")
```



(a) (b)

Figure 2: (a)Hamming window - spectrum for Pre-Emphasized sound (b)Combined spectrum plot for Pre-emphasized and original sound

1.3 Question 3

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.

Solution :

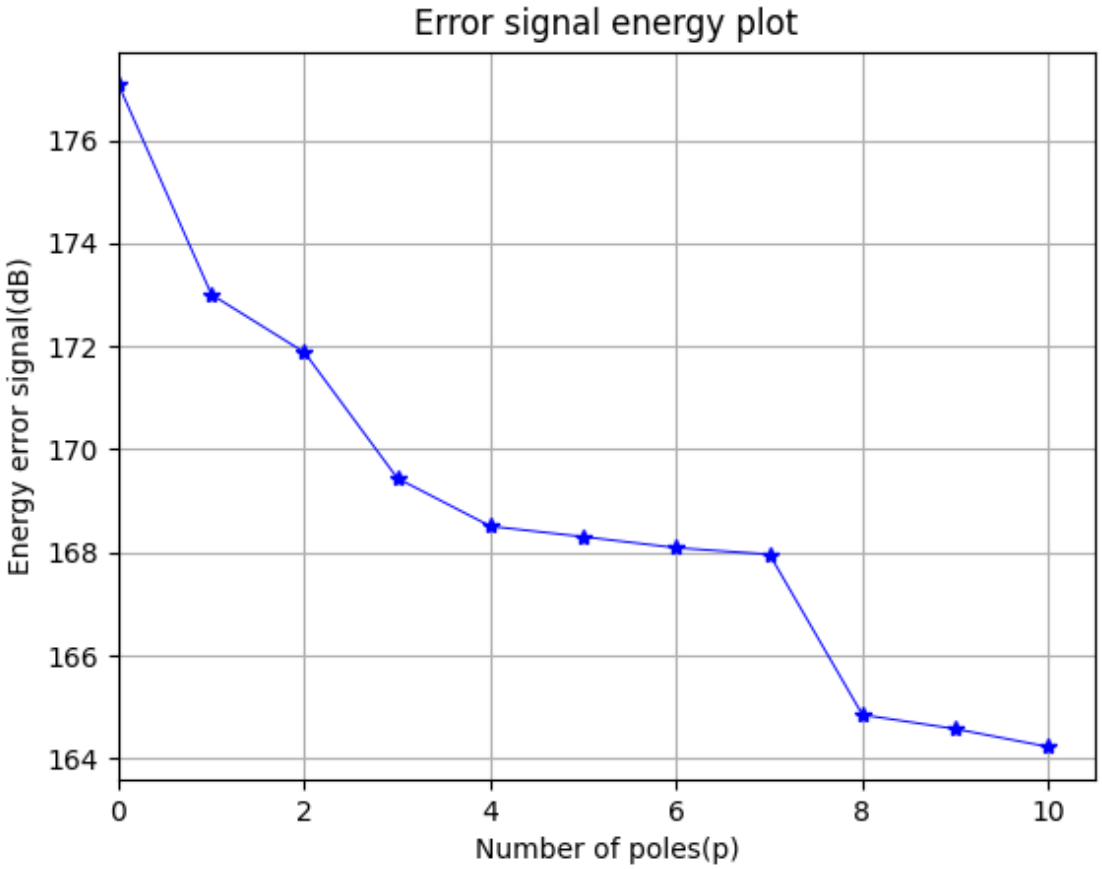
Code :

Levinson Durbin Function :

```
def LPRecursion(window_signal,p):
    r= np.correlate(window_signal_pre,window_signal_pre,mode='full')
    r= r[-len(window_signal_pre):]  # keeping only the positive coefficients since
    correlation is symmetric

    e = np.zeros(p+1)
    a = np.zeros((p+1,p+1))
    g = np.zeros(p+1)

    e[0] = r[0]
    g[0] = np.sqrt(e[0])
```

```
    a[1][0] = 1

    k=r[1]/e[0]
    a[1][1] = k
    e[1]=(1-k**2)*e[0]
    g[1] = np.sqrt(e[1])

    for i in range(2,p+1):
        temp = 0
        for j in range(1,i):
            temp+=a[i-1][j] *r[i-j]
        k = (r[i] - temp)/e[i-1]
        a[i][i] = k
        for j in range(1,i):
            a[i][j] = a[i-1][j] - k*a[i-1][i-j]
        e[i] = (1-k**2)*e[i-1]
        g[i] = np.sqrt(e[i])
        a[i][0]=1

    return g,e,a
```

Error spectrum :



(a)

Figure 3: (a)Error Signal Energy with number of ploes

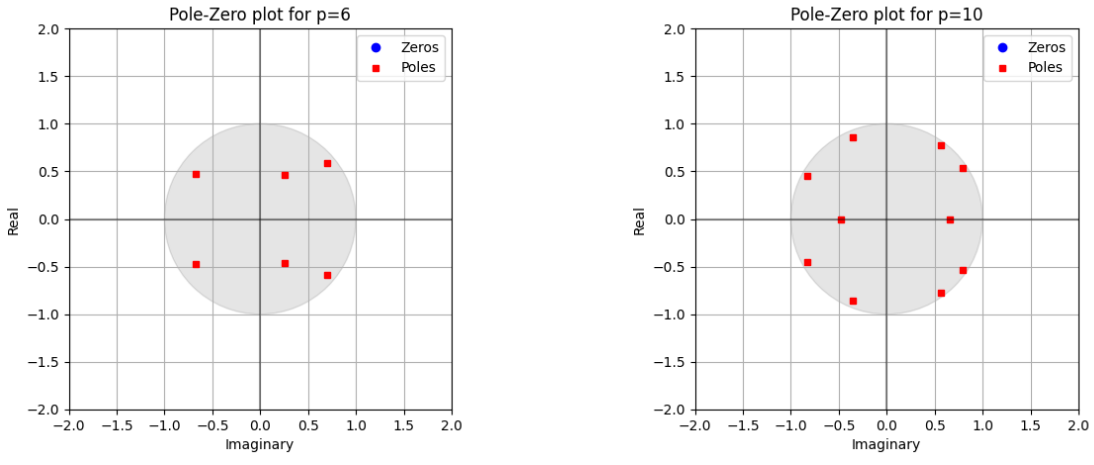
1.4 Question 4

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

Solution : Code :
Pole Zero Plot Function :

```
def poleZeroPlot(b,a,title_):

    poles = np.zeros_like(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.title(title_)
    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.grid()

    plt.savefig('outputs/'+str(title_))
```

Error spectrum :



(a) (b)

Figure 4: (a)Pole-Zero Plot for P=6 (b)Pole-Zero Plot for P=10

Comments :

1. The poles observed are conjugate symmetric, while there are no zeros.(all pole system)
2. The number of formants can be decided on the basis of number of poles (2 poles correspond to a single formant frequency). Thus for a system with 6 poles, we expect 3 peaks (formant frequency) while for 10 poles, we expect at most 5 peaks in the spectrum.
3. The location of the poles will determine the formant frequency and the bandwidth of the peak.
4. Since there are no zeros, 'valleys' in the spectrum plot will not be created explicitly. However, if we have several poles close to the edge of unit circle, this will create peaks of high amplitude, and hence valleys will be created relative to these pole locations.

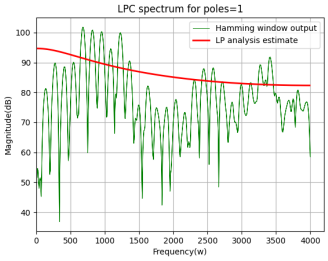
1.5 Question 5

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.
Solution :

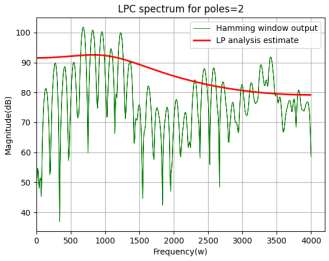
Code :
Pole LPC Spectrum Magnitude :

```
def plot_LPC(p,a,g,dft_pre,fs,freq):
    for i in range(1,p+1):
        poles = np.zeros_like(a[i])
        poles[0] = a[i][0]
        poles[1:] = -a[i][1:len(a[i])+1]
        w,h = freqz(g[i],poles)

        plt.figure()
        plt.title("LPC spectrum for poles="+str(i))
        plt.plot(abs(freq),20*log10(abs(dft_pre)),'g',linewidth=0.7)
        plt.plot(w*fs/(2*np.pi),20*log10(abs(h)),'r',linewidth=2)
        plt.ylabel("Magnitude(dB)")
        plt.xlabel("Frequency(w)")
        plt.xlim(xmin=0)
        plt.grid("True")
        plt.legend(['Hamming window output' , 'LP analysis estimate'])
        plt.savefig('outputs/LPCSpectrum_poles_'+str(i)+".png")
```

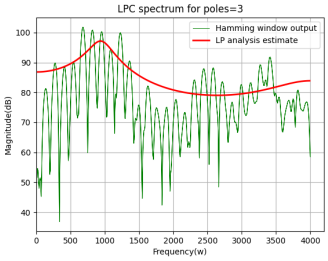
LPC Spectrums :



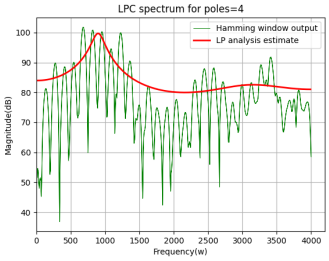
(a)



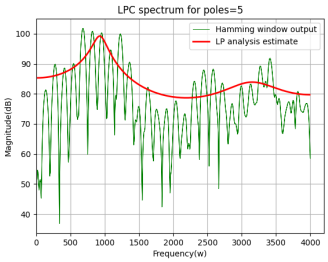
(b)



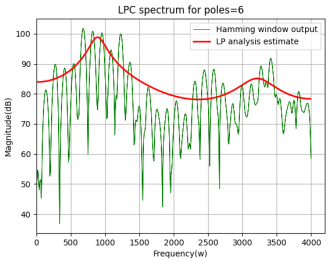
(c)



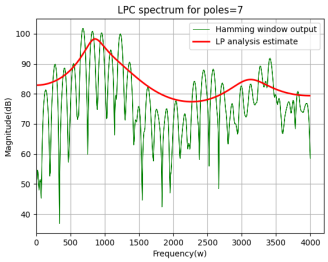
(d)



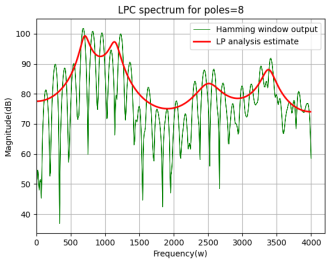
(e)



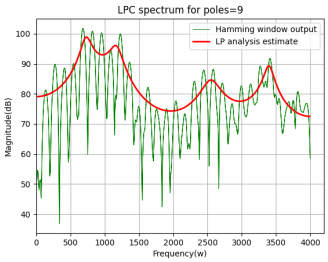
(f)



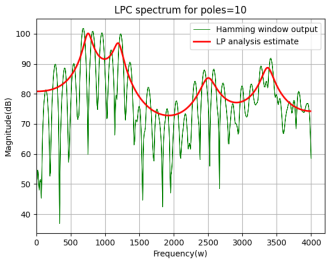
(g)



(h)



(i)



(j)

Figure 5: (a)P=1 (b)P=2 (c)P=3 (d)P=4 (e)P=5 (f)P=6 (g)P=7 (h)P=8 (i)P=9 (j)P=10

Comments:

1. The short time Fourier spectrum of the input signal computed earlier has 4 clear peak regions which may be the formants of the particular sound.
2. For p=1..4 only one clear peak is obtained.
3. Hence it is expected that a LP analysis of order >=8 should give a good approximation of the signal envelope.
4. As we increase the order p , we can clearly see that the approximation improves and the number of peaks increases with every 2nd order value.
5. This is expected as 2 poles correspond to a single formant frequency .
6. The approximation is almost perfect from p=8 on wards, however it improves further as we go to p=10, where we can see that the peaks are sharper (have a lower bandwidth) as compared to p=8 or p=9.
7. The value of Gain G decreases with increase in the order as per the following equation -

$$G = \sqrt{ACF[0] - \sum_{k=1}^{p} \alpha_k ACF[k]} \tag{3}$$

Same has been obtained as per the code as well.

1.6 Question 6

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.

Solution :

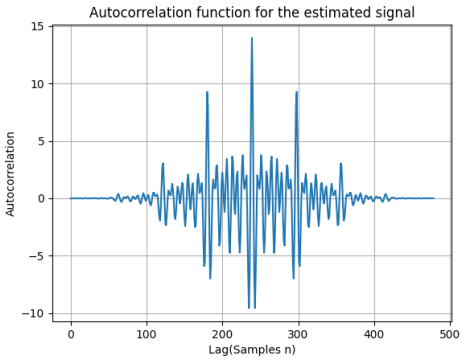
Code :

Inverse Filtering using difference equation :

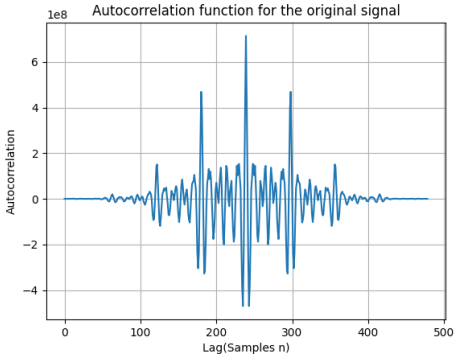
```
    input_signal = window_signal_pre
    zeros_filter = a[order][:]
    poles = g[order]
    output = np.zeros(len(input_signal))

    for i in range(len(input_signal)):
        temp = 0
        for j in range(0,order):
            if i-j>=0:
                output[i]+=(-zeros_filter[j]*input_signal[i-j])
        output[i] = output[i]/poles

    acf = np.correlate(output,output,mode="full")
    original_acf =np.correlate(window_signal_pre,window_signal_pre,mode="full")
    plot_acf(acf,'Autocorrelation function for the estimated signal','acf_estimated')
    plot_acf(original_acf,'Autocorrelation function for the estimated signal','
    acf_original')
```

```
def plot_acf(acf,title_,fig_name):
    plt.figure()
    plt.plot(acf)
    plt.xlabel('Lag(Samples n)')
    plt.ylabel('Autocorrelation')
    plt.grid()
    plt.title(title_)
    plt.savefig('outputs/'+fig_name+'.png')
```

ACF Plots :



(a)



(b)

Figure 6: LPC Spectrum Plots (a)Residual signal ACF (b)Original signal ACF

Comments :

1. By plotting the ACF of the residual waveform, we can clearly see peaks which are periodic . The difference between the location of these peaks will give us the pitch period of the sound.

2. The estimated pitch comes out to be $136Hz$.The same value has been used in the next question to re-synthesize the sound.

3. The ACF plot of the original signal and the residual signal are almost identical. The only difference which can be seen is in the magnitude of the ACF value obtained.

4. The estimated pitch was calculated by manually finding the distance between the peaks as obtained in the plot. ($F_0 = 136Hz$)

1.7 Question 7

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. 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. Comment on the similarity with the original sound. What would be a good application for this analysis-and-synthesis system, and how exactly does it help?

Solution :

1. First we generate impulse train of the estimated fundamental frequency (136 Hz)
2. Next we take the 10th order filter coefficients and perform the filtering using difference equation
3. The output waveform is de-emphasized and then normalized to get the final output
4. The output wav file is stored in the outputs folder

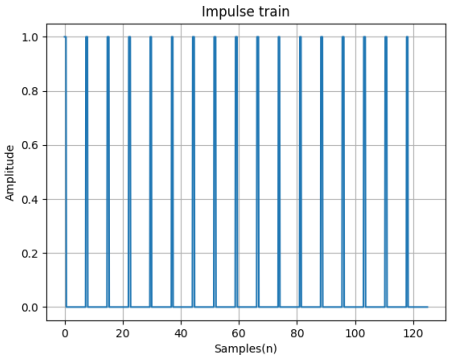
Code :

Reconstructing the signal :

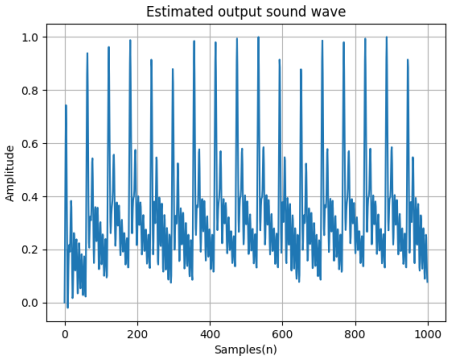
```
    estimated_f = 136
    t = np.linspace(0, 300, int(300*fs/1000), endpoint=False)
    sig = (signal.square(2 * np.pi *(estimated_f/1000)* t[0:1000],duty=0.08)+1)/2   # Impulse Train of estimated frequency
    plt.figure()
    plt.plot(t[0:1000], sig)
    plt.title('Impulse train')
    plt.xlabel('Samples(n)')
    plt.ylabel('Amplitude')
    plt.grid()
    plt.savefig('outputs/impulse_train.png')


    order =10
    output=np.zeros(sig.shape)

    for i in range(len(sig)):  #Filtering using difference equation order 10
        output[i] = g[order]*sig[i]
        for j in range(0,order):
            if i-j>=0:
                output[i]+=a[order][j]*output[i-j]

    output_pre = np.zeros(sig.shape)
    alpha=0.98
    for i in range(1,len(output)):   #De-emphasis
        output_pre[i] = output[i] + alpha*output_pre[i-1]

    maxi = np.max(output_pre)
    output_pre = output_pre/maxi

    plt.figure()
    plt.plot(output_pre)
    plt.title('Estimated output sound wave')
    plt.xlabel('Samples(n)')
```

```
    plt.ylabel('Amplitude')
    plt.grid()
    plt.savefig('outputs/estimated_output.png')
    wav.write('outputs/result.wav',fs,output_pre)
```

Output Plots :



(a)



(b)

Figure 7: LPC Spectrum Plots (a)Impulse train input F0 = 136 Hz (b)Estimated output signal waveform

Comments :

1. We perform de-emphasis and the normalize the final output to get the reconstructed sound. De-emphasis is done to revert the pre-emphasis done initially. Normalizing the output amplitude gives a more realistic sound, which is close to the original sound we started with.

2. An impulse train is created having fundamental frequency of 136 Hz as it was calculated earlier. This is passed through order 10 system which is obtained earlier.

3. The computation is done using difference equation method.

4. The final reconstructed sound file is stored in the outputs folder. It is very close to original "aa" sound wave.

5. One of the applications of such AR models is in channel estimation in the field of wireless communication. Other possibility. In the context of speech this can be used for audio compression. As we need to transmit/store only the values for G,p,a and F_0 to reconstruct the original signal at the receiver end, this results in a huge decrease in the number of bits required for transmission.