Assignment 2 Speech Processing

Aditya Khanna 17D070036
Faculty mentor : Prof. Preeti Rao

2nd November 2020

1 Question

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

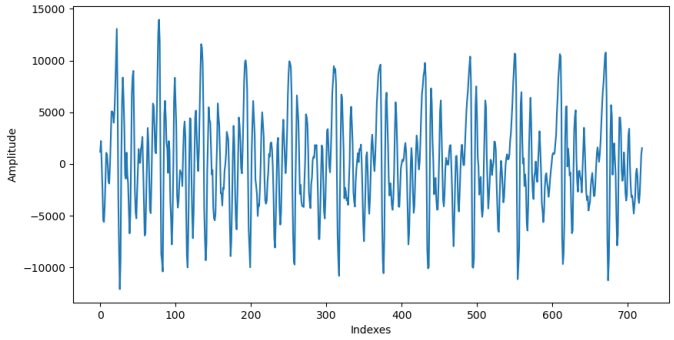


Figure 1: Without Pre Emphasis

```
import scipy
from scipy import signal
from os.path import dirname, join as pjoin
from scipy.io import wavfile
import scipy.io
import numpy as np
import math
from scipy.fft import fft
import matplotlib.pyplot as plt
file = scipy.io.wavfile.read('aa.wav')
inp = file[1]
length = len(file[1])
print(len(inp), "No of points in given sample")
print(file[0],"Sampling Freq")
plt.figure(figsize=(10, 5))
plt.xlabel('Indexes')
plt.ylabel('Amplitude')
plt.plot(inp)

```

1.1

1. Apply pre-emphasis to the signal(aplha = 0.96)

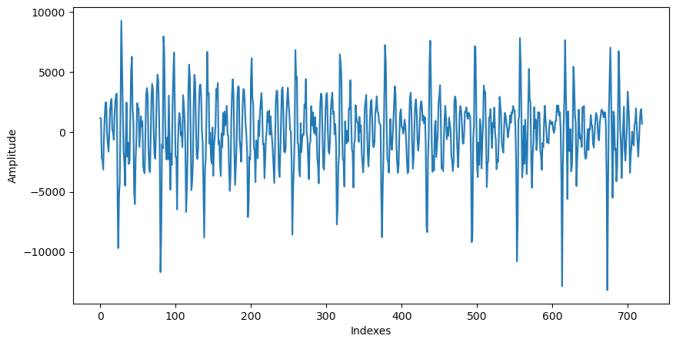


Figure 2: With Pre Emphasis

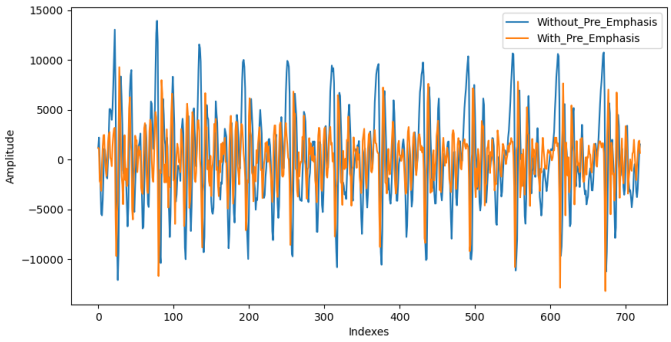


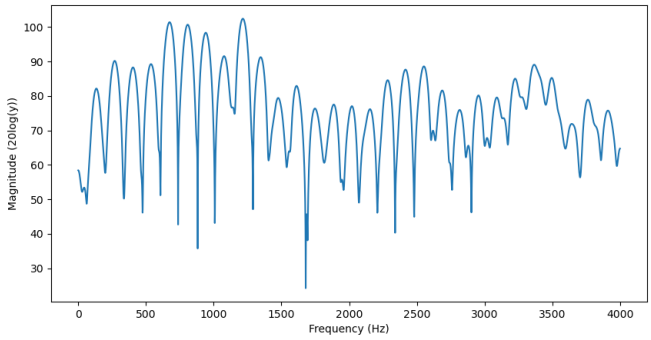
Figure 3: Comparison

```python
# Premphasis of the signal
y=[]
alpha = 0.96
y.append(inp[0])
for i in range(1,length):
  y.append(inp[i] - alpha*inp[i-1])
print(len(y))
plt.figure(figsize=(10, 5))
plt.xlabel('Indexes')
plt.ylabel('Amplitude')
plt.plot(y)

```

1.2

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




```python
# Narrowband Magnitude Spectrum Slice (0 to 4000 Hz)
Window_length = 0.03
F_s = 8000
y_new = (y[240:480]) * np.hamming(int(Window_length*F_s)) # Middle of 0,720 is is 240:480
dft = np.fft.fft(y_new, F_s)
x_axis = np.arange(0, F_s/2 , 1)
plt.figure(figsize=(10, 5))
plt.plot(20*np.log10(dft[:int(F_s/2)]))
plt.xlabel('Frequency (Hz)')
plt.ylabel('Magnitude (20log(y))')
plt.show()

```

1.3

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.

```
def acc(inp,k):
    sum=0
    y_new = inp * np.hamming(int(len(inp)))
    for i in range(len(inp)-k):
        sum+=y_new[i]*y_new[i+k]
    return sum;
print("Coefficient for 2",acc(y[240:480],2))
print("Coefficient for 4",acc(y[240:480],4))
print("Coefficient for 6",acc(y[240:480],6))
print("Coefficient for 8",acc(y[240:480],8))
print("Coefficient for 10",acc(y[240:480],10))

```

Auto Correlation Coefficient for 2 = 67303413.96184503
Auto Correlation Coefficient for 4 = -436465215.6476679
Auto Correlation Coefficient for 6 = -105817984.85330452
Auto Correlation Coefficient for 8 = 51911193.40986656
Auto Correlation Coefficient for 10 = 36969773.46101895

```
def levinsondurbin(inp,p):
    a=[]
    e=[]
    E_0=acc(inp,0)
    k_1= acc(inp,1)/E_0
    a.append([k_1])
    e.append((1-k_1*k_1)*E_0)
    for i in range(2,p+1):
        r_i=acc(inp,i)
        sum=0
        for j in range(1,i):
            sum+=a[i-2][j-1]*acc(inp,i-j)
        k_i=(r_i - sum)/ e[i-2]
        rand=[]
        for j in range(1,i):
            rand.append(a[i-2][j-1] - k_i * a[i-2][i-j-1])
        rand.append(k_i)
        a.append(rand)
        e.append((1 - k_i*k_i)*e[i-2])
    return a,e
# a_2,e_2= levinsondurbin(y[240:480],2)
# a_4,e_4= levinsondurbin(y[240:480],4)
# a_6,e_6= levinsondurbin(y[240:480],6)
# a_8,e_8= levinsondurbin(y[240:480],8)
a_10,e_10= levinsondurbin(y[240:480],10)
plt.figure(figsize=(10, 5))
plt.plot(e_10)
plt.xlabel('p')
plt.ylabel(' Error signal energy (i.e. square of gain) ')
plt.show()

```

LP Coefficients for different values of P

```
P=1; [0.6367718245962932]
P=2; [0.9645477609543258, -0.5147462932516513]
P=3; [0.7516906966224371, -0.1158880450305983, -0.41351840143864077]
P=4;[0.6832214815281074, -0.13507646192923248, -0.2890555678322258, -0.16557719041310784]
P=5;[0.7080240101491009, -0.0917775692718005, -0.2688218761947265, -0.2679199123049333,
    0.1497943560892195]
P=6;[0.7363080120582046, -0.14236590590182785, -0.3195805206323835, -0.2852492497530704,
    0.283482653683161, -0.18881887574093487]
P=7;[0.6999537363193167, -0.08778552288265778, -0.37450103553928304, -0.3467797419893839,
    0.25607220873340547, -0.04705368201187782, -0.19253517740867745]
P=8;[0.5881573008589365, -0.11510745955425258, -0.22581153088790631, -0.5481389926264375,
    0.03861646161741861, -0.0980267492476653, 0.2138961758685223, -0.5806545949942427]
P=9;[0.7550729887265398, -0.1765943187561474, -0.19763264033721437, -0.5592397284329771,
    0.19618517422220585, -0.03311468735833305, 0.24698510852457217, -0.7497270199669508,
    0.28746123651920524]
P=10;[0.7110376083400064, -0.06174573983096372, -0.23546759622686578, -0.5541669823755868,
    0.166132117755596, 0.05255367776456771, 0.2772598982646586, -0.7226750316895395,
    0.1717937236371738, 0.1531871946275144]

```

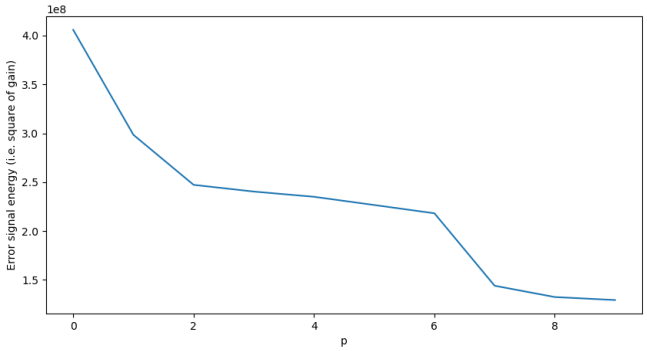


Figure 4: error signal energy (i.e. square of gain) vs p.

Error value variations with P

```
Error when P=1 402344064.7394002
Error when P=2 295737473.9976803
Error when P=3 245167114.65425017
Error when P=4 238445660.60496286
Error when P=5 233095333.6270696
Error when P=6 224784886.43265897
Error when P=7 216452156.8769501
Error when P=8 143473199.89683604
Error when P=9 131617440.87664083
Error when P=10 128528864.33923104
```

1.4

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

1.4.1 P=6

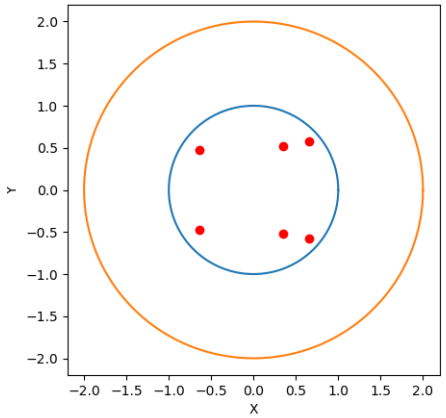


Figure 5: P=6

```
aa=[1]
for i in a_10[5]:
    aa.append(-1* i)
poles_6 = np.roots(aa)
X = [x.real for x in poles_6]
Y = [x.imag for x in poles_6]
r0=1
r1=2
```

```
plt.figure(figsize=(5, 5))
plt.xlabel('X')
plt.ylabel('Y')
plt.scatter(X,Y, color='red')
theta = np.arange(0, np.pi*2, 0.01)
plt.plot(r0 * np.cos(theta), r0 * np.sin(theta))
plt.plot(r1 * np.cos(theta), r1 * np.sin(theta))
plt.show()
print(poles_6)
```

Coordinates of Poles in P=6

```
-0.63742054+0.47891212j
-0.63742054-0.47891212j
0.65982201+0.58240455j
0.65982201-0.58240455j
0.34575253+0.51376466j
0.34575253-0.51376466j
```

For p=6, there are 4 poles and all are inside the unit radius circle. None of the poles are real, and contain non zero imaginary part.The poles occur as conjugates i.e symmetrical about the x axis.

1.4.2 P=10

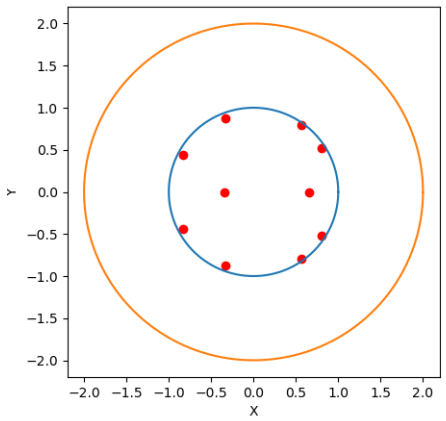


Figure 6: P=10

```
# Poles for p=10
aaa=[1]
for i in a_10[9]:
    aaa.append(-1* i)
poles_10 = np.roots(aaa)
X = [x.real for x in poles_10]
Y = [x.imag for x in poles_10]
r0=1
r1=2
plt.figure(figsize=(5, 5))
plt.xlabel('X')
plt.ylabel('Y')
plt.scatter(X,Y, color='red')
theta = np.arange(0, np.pi*2, 0.01)
plt.plot(r0 * np.cos(theta), r0 * np.sin(theta))
plt.plot(r1 * np.cos(theta), r1 * np.sin(theta))
plt.show()
print(poles_10)
```

Coordinates of Poles in P=10

```
0.56120875+0.78976252j
0.56120875-0.78976252j
0.80425681+0.52217128j
0.80425681-0.52217128j
0.66327132+0.j
```

```
-0.33552838+0.87509787j
-0.33552838-0.87509787j
-0.83638979+0.44522166j
-0.83638979-0.44522166j
-0.33932847+0.j
```

For p=10, there are 10 poles out of which 2 poles lie on the x axis i.e the imaginary part is 0. The remaining 8 poles are all imgainary. Thus for P=10, we will get 4 peaks as 2 poles will be used for modelling.The poles occur as conjugates i.e symmetrical about the x axis.

1.5

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

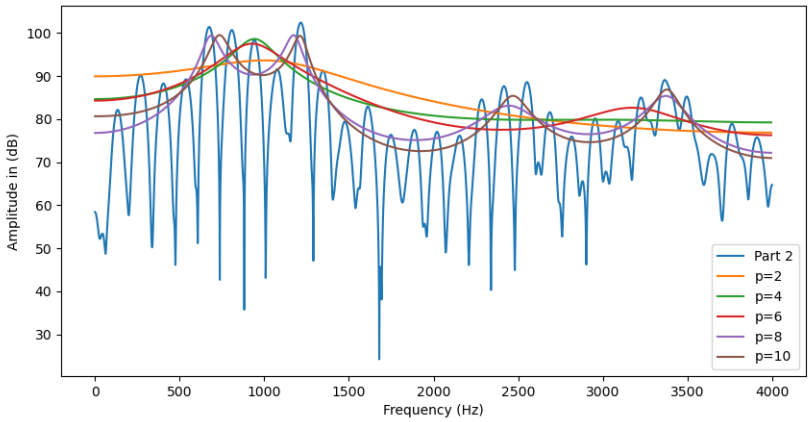


Figure 7: P=10

Both P=2 and P=4 contain 1 formant,For P=2, the bandwidth is very high and for P=4, the bandwidth decreases from P=2, but is still high.
For P=6, there are 2 formants with one with high bandwidth and one with moderate like p=4 bandwidth.P=8 and P=10 both have 4 formants, with 2 poles of P=10 being real, thus leading to 4 formants. All these bandwidths are relatively low. The spectral envelopes as compare to short-time magnitude spectrum in part 2, follows the peaks trends, and a further increase in p, will lead to more peaks, Thus we must tune the p to around 10-12 after which we would have no benifit. The P values tries to envelope the spectrum and represent it really well, and as the value increases, from 2-10 the envelope becomes better and better and their is better overlap too but after this the peaks will increase and not lead to much benifit.

```
def LPC_spectrum(p):
    rand=[1]
    for i in a_10[p-1]:
        rand.append(-1* i)
    den = rand
    num = math.sqrt(e_10[p-1])
    w, h = scipy.signal.freqz(num, den)
    plt.plot(w*4000/3.14, 20*np.log10(abs(h)), label='p=' + str(p))
    plt.xlabel('Frequency (Hz)')
    plt.ylabel('Amplitude in (dB)')
    plt.figure(figsize=(10, 5))
plt.plot(20*np.log10(dft[:int(F_s/2)]),label= "Part 2")
LPC_spectrum(2)
LPC_spectrum(4)
LPC_spectrum(6)
LPC_spectrum(8)
LPC_spectrum(10)
plt.show()
```

1.6

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.

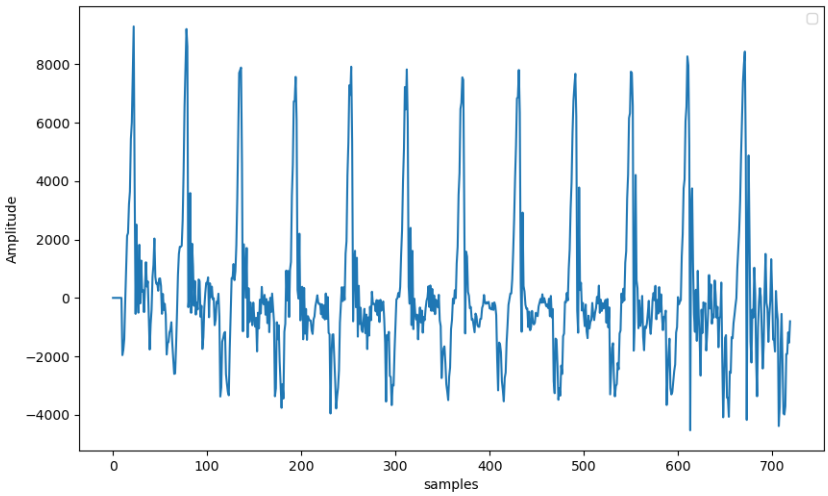


Figure 8: P=10

The error terms seems to be impulsive and jumps really fast. Error term while decaying, decays slowly between the peaks. We can measure the pitch period of the voiced sound from the residual waveform using the acf function, due to the impulsive behavior

```
error= []
for i in range(0,10):
    error.append(0)
for i in range(10,720):
    error.append(inp[i])
    for j in range(1,11):
        error[i]-=a_10[9][j-1]*inp[i-j]
plt.figure(figsize=(10, 6))
plt.plot(error)
plt.xlabel("samples")
plt.ylabel("Amplitude")
plt.legend()
plt.show()
```

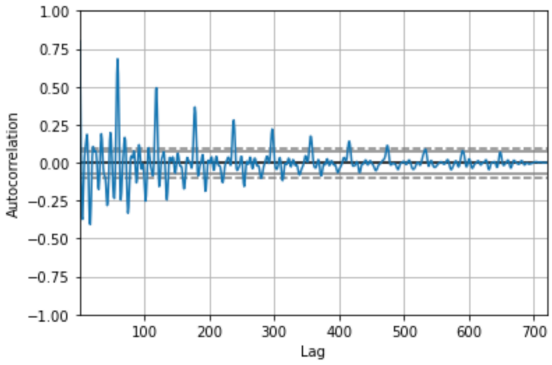


Figure 9: ACF of Normalized Input signal

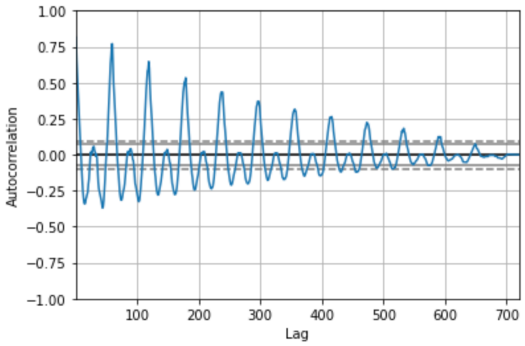


Figure 10: ACF of Residual signal

To find the pitch of the signal we must subtract the difference in time of the 2 consecutive peaks(Initial Peaks), whilst ignoring the small peaks. Analytically measuring,There is a difference of about 65 samples in between the peaks, so 8000/65 is 123 Hz.

Since the residual part is really accurate and very less noisy, it is easier to calculate from the residual than the original signal. The plot of the input signal is also periodic with a decaying envelope and decays much faster than the residual signal. The sharpness in the initial signal plot is very high and its very difficult to accurate peaks from this graph.

```
x = pd.plotting.autocorrelation_plot(error)
plot(x)
x = pd.plotting.autocorrelation_plot(inp/np.amax(inp))
plot(x)
```