Assignment 2: Linear Predictive Analysis

Pranav Sankhe— 150070009

4/10/2018

1 Question

Synthesized vowel: Consider the synthesized vowel /a/ (formants: 730, 1090, 2440 Hz; bandwidths: 50 Hz) at two fundamental frequencies: 120 Hz, 300 Hz. Sampling rate = 8 kHz. Using a 30 ms Hamming window, implement LP analysis on a single segment using LP orders 2, 4, 6, 8, 10 using the Levinson algorithm. Compute the gain, and plot the LP spectrum magnitude (i.e. the dB magnitude frequency response of the estimated all-pole filter) for each order "p". Superimpose each plot on the original 6-pole spectral envelope with the discrete harmonic components shown with vertical lines. Comment on the characteristics of the spectral approximations of different orders.

1.1 Answer

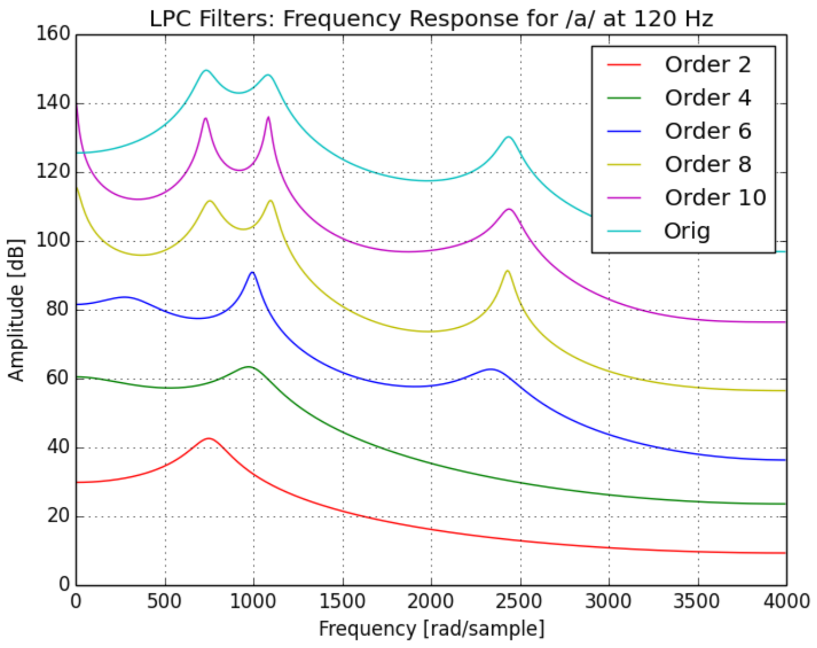

Figure 1: \a\at 120 Hz

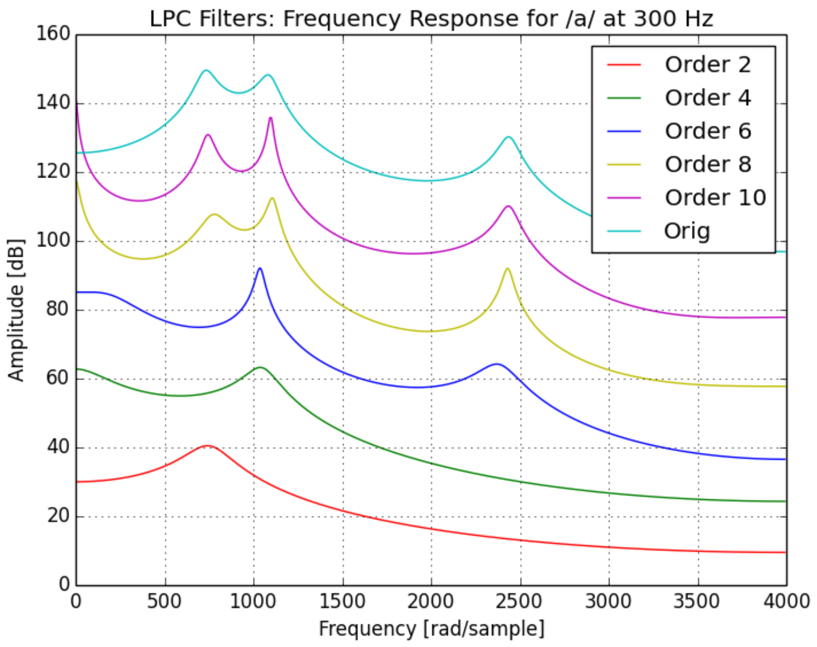

Figure 2: \a\at 300 Hz

It is evident from the graphs that the approximation of the real spectrum keeps on getting better as we increase the order at both the signal frequencies. This is expected since as we increase the order, the LP spectrum is able to capture more and more details of the original spectrum. Also, as we increase the number of poles (order) we observe that there's an increase in the sharpness of the peaks.

File containing all the paramaters:

```
formant_freq = [730, 1090, 2440]
formant_bw = [100, 100, 100]
samp_freq = 8000.0
sig_freq = 300
time_length = 0.5
win_size = 30.0
dft_len = 1024
filter_order= [2, 4, 6, 8, 10]
```

Listing 1: hparams.py

main file:

```
import librosa
import numpy as np
from matplotlib import pyplot as plt
import pylab
from scipy import signal
from scipy.io.wavfile import write
import hparams
import sys
```

```python
from math import pi


def autocorr(x):
    result = np.correlate(x, x, mode='full')
    return result[int(result.size/2):]


sig_freq = hparams.sig_freq

y, samp_freq = librosa.load( str(sig_freq) + '.wav',sr=None)
win_size = hparams.win_size/1000.0
num_samples = int(samp_freq*win_size)
window = y[:num_samples]*np.hamming(num_samples)

fig1 = plt.figure()
plt.title('LPC Filters: Frequency Response for /a/ at '+str(
    sig_freq)+' Hz')
plt.ylabel('Amplitude [dB]')
plt.xlabel('Frequency [rad/sample]')

colors = {2: 'r', 4: 'g', 6: 'b', 8: 'y', 10: 'm'}
orders = hparams.filter_order

for order in orders:

    R = autocorr(window)
    error = np.zeros(order+1)
    error[0] = R[0]
    G = np.zeros(order + 1)

    coeffs = np.zeros(order+1)
    dummy_coeffs = np.zeros(order + 1)


    for i in range(1, order +1):
        reflec_coeffs = 0
        dummy_coeffs[1:len(coeffs)] = coeffs[1:len(coeffs)]

        for j in range(1, i):
            reflec_coeffs = reflec_coeffs + dummy_coeffs[j]*R[i-j]
        reflec_coeffs = (R[i] - reflec_coeffs)/error[i-1]

        coeffs[i] = reflec_coeffs

        for j in range(1, i):
            coeffs[j] = dummy_coeffs[j] - reflec_coeffs*
    dummy_coeffs[i-j]

        error[i] = (1-np.square(reflec_coeffs))*error[i-1]

    coeffs[0] = 1.0
    coeffs[1:len(coeffs)] = -coeffs[1:len(coeffs)]
    num_coeffs = np.zeros(coeffs.shape)
    num_coeffs[0] = 1
    G[i] = np.sqrt(error[i])
    w, h = signal.freqz(num_coeffs, coeffs)

    plt.plot(samp_freq*w/(2*np.pi), 10*order + 20 * np.log10(abs(h)
    ), colors[order])

formant_freq = hparams.formant_freq
```

```python
formant_bw = hparams.formant_bw

r = np.exp(np.multiply(-np.pi, formant_bw)/samp_freq)
theta = 2*np.multiply(np.pi, formant_freq)/samp_freq

poles = np.concatenate([r * np.exp(1j*theta), r * np.exp(-1j*theta)
    ])
zeros = np.zeros(poles.shape, poles.dtype)

b, a = signal.zpk2tf(zeros, poles, 1)

wf, hf = signal.freqz(b, a)

plt.plot(samp_freq*wf/(2*pi), 120 + 20 * np.log10(abs(hf)), 'c')
plt.legend(['Order 2', 'Order 4', 'Order 6', 'Order 8', 'Order 10',
    'Orig'])
plt.grid()

plt.show()
```

Listing 2: q1.py

2 Question 2

Natural speech: Consider the speech signal in *machali.wav* (male voice), sampled at 8 kHz. Consider the following signal segments in the final word "pani": (1) \a\(first half); (2) \n\; (3) \I\and (4) \s\in the word *uska.*

Use PRAAT to extract the above segments to separate .wav files for further analyses as below. (Note: for \s\, 16 kHz sampled audio is better.)

Compute and plot the narrowband spectrum using a Hamming window of duration = 30 ms before and after pre-emphasis.

Using a 30 ms Hamming window centered in the segment of the waveform (pre-emphasised for the voiced sounds):

Compute the autocorrelation coefficients required for LPC calculation at various p = $4, 6, 8, 10, 12, 20$. Use the Levinson algorithm to compute the LP coefficients from the autocorrelation coefficients. Show the pole-zero plots of the estimated all-pole filter for p=$6, 10$.

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". Superimpose each plot on the narrowband dB magnitude spectrum of part 1 (after pre-emphasis). Comment on the characteristics of the spectra.

Plot error signal energy (i.e. square of gain) vs p.

2.1 Answer

Pre-Emphasis Glotal shaping and radiation can be modelled by an one pole filter. We remove it by passing the signal through a pre emphasis filter. A pre-emphasis filter is used to counteract glottal roll-off in voiced speech. This helps to model the environment.

Note that in the plots, the original spectra is in blue and the pre-emphasized spectra is in green.

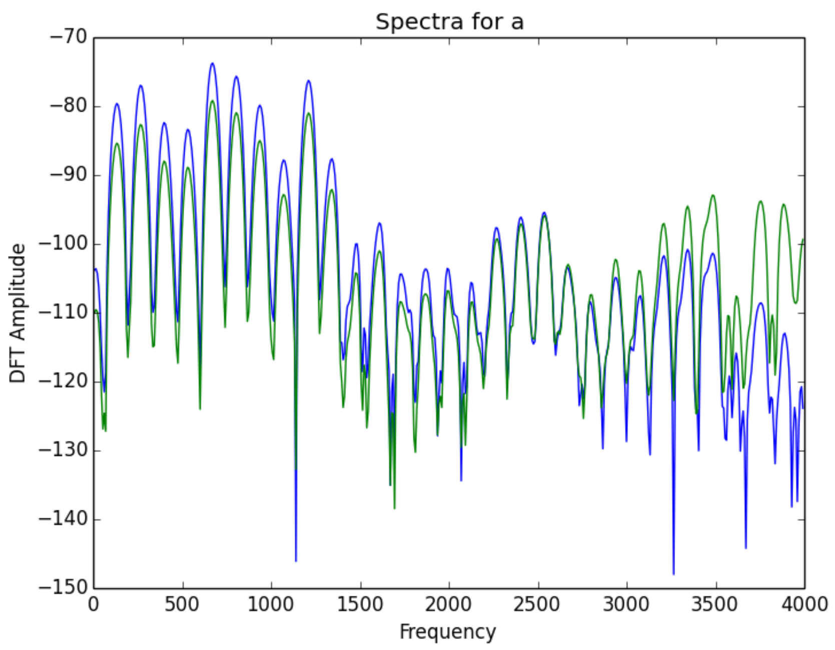

Figure 3: \a\from the natural recording of *machali*

Poles and Zeros

```python
win_size = 30.0
dft_length = 1024
files = ['machali_male_8k_a.wav', 'machali_male_8k_n.wav', '
    machali_male_8k_i.wav', 'machali_male_16k_s.wav']
colors = {12: 'r', 4: 'g', 6: 'b', 8: 'y', 10: 'm', 20: 'k'}
displacements = {12: 80, 4: 0, 6: 20, 8: 40, 10: 60, 20: 100}
orders = [4, 6, 8, 10, 12, 20]
file_index = 0
```

Listing 3: hparams.py

```python
import librosa
from scipy import signal
import numpy as np
from math import pi
import matplotlib.pyplot as plt
import hparams

def autocorr(x):
    result = np.correlate(x, x, mode='full')
    return result[int(result.size/2):]


files = hparams.files
file_index = hparams.file_index
colors = hparams.colors
displacements = hparams.displacements
orders = hparams.orders

```

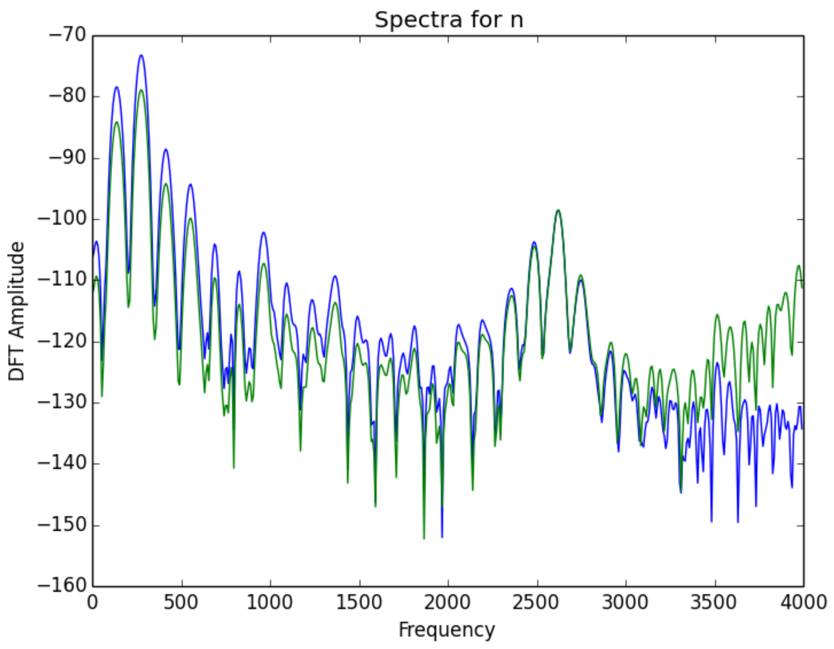

Figure 4: \n\from the natural recording of *machali*

```

y, samp_freq = librosa.load(files[file_index],sr=None)
y = y/32768
win_size = hparams.win_size/1000.0
num_samples = int(samp_freq*win_size)

window = y[int((len(y)-num_samples)/2):int((len(y) + num_samples)
    /2)]*np.hamming(num_samples)

dft = np.fft.fft(window, hparams.dft_length)
freq = np.fft.fftfreq(dft.shape[-1], 1/float(samp_freq))
fig1 = plt.figure()
plt.plot(freq[:int(len(freq)/2)], 20*np.log10(np.abs(dft[:int(len(
    dft)/2)])))
plt.ylabel('DFT Amplitude')
plt.xlabel('Frequency')
plt.title('Spectra for '+files[file_index][-5:-4])


for i in range(1, len(window)):
    window[i] = window[i] - 15.0/16.0 * window[i-1]

dft = np.fft.fft(window, hparams.dft_length)
freq = np.fft.fftfreq(dft.shape[-1], 1/float(samp_freq))
plt.plot(freq[:int(len(freq)/2)], 20*np.log10(np.abs(dft[:int(len(
    dft)/2)])), 'g')

orders = np.multiply(orders,(samp_freq/8000))
for order in orders:
    order = int(order)
```

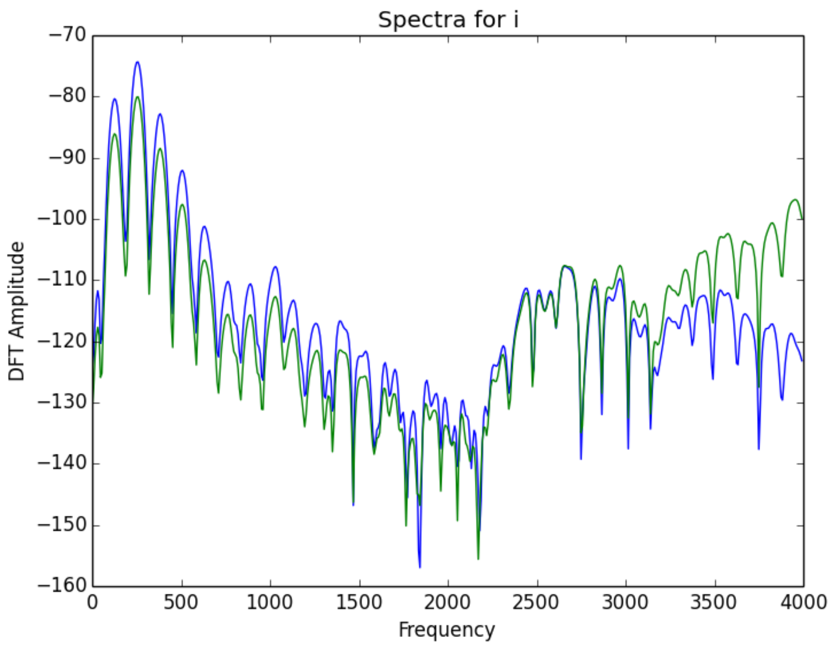

Figure 5: \I\from the natural recording of *machali*

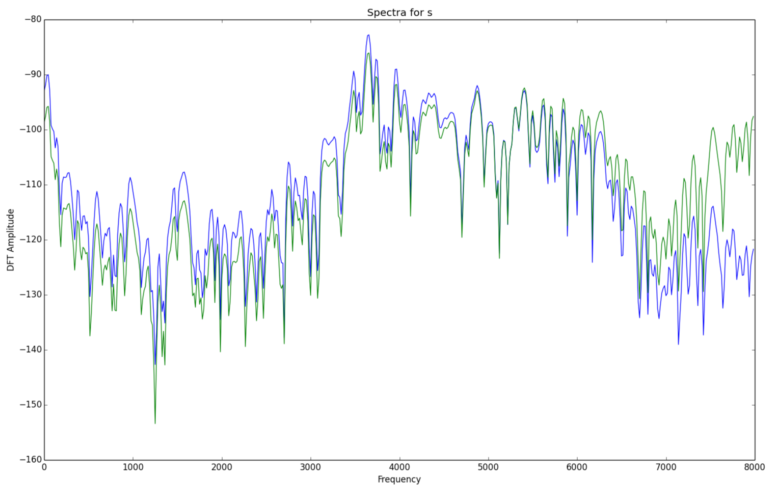

Figure 6: \s\from the natural recording of *machali*

```

    R = autocorr(window)
     error = np.zeros(order+1)
     error[0] = R[0]
    G = np.zeros(order + 1)
```

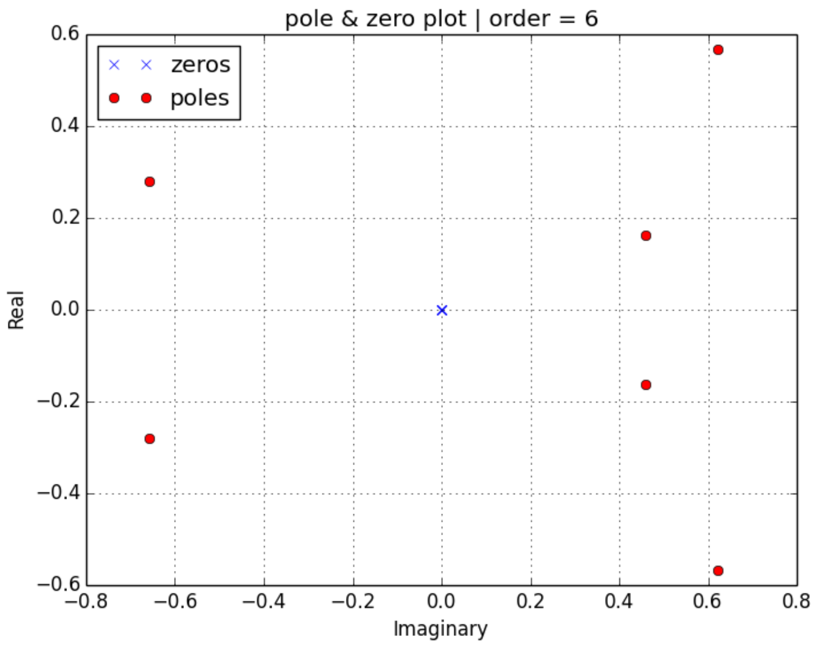

Figure 7: Poles and zeros when order = 6 for \a\from the natural recording of *machali*

```

    coeffs = np.zeros(order+1)
    dummy_coeffs = np.zeros(order + 1)


    for i in range(1, order +1):
        reflec_coeffs = 0
        dummy_coeffs[1:len(coeffs)] = coeffs[1:len(coeffs)]

        for j in range(1, i):
            reflec_coeffs = reflec_coeffs + dummy_coeffs[j]*R[i-j]
        reflec_coeffs = (R[i] - reflec_coeffs)/error[i - 1]

        coeffs[i] = reflec_coeffs

        for j in range(1, i):
            coeffs[j] = dummy_coeffs[j] - reflec_coeffs*
    dummy_coeffs[i-j]

        error[i] = (1-np.square(reflec_coeffs))*error[i-1]
        G[i] = np.sqrt(error[i])
    coeffs[0] = 1.0
    coeffs[1:len(coeffs)] = -coeffs[1:len(coeffs)]
    num_coeffs = np.zeros(coeffs.shape)
    num_coeffs[0] = 1

    w, h = signal.freqz(num_coeffs, coeffs)

    if file_index == 3:
```

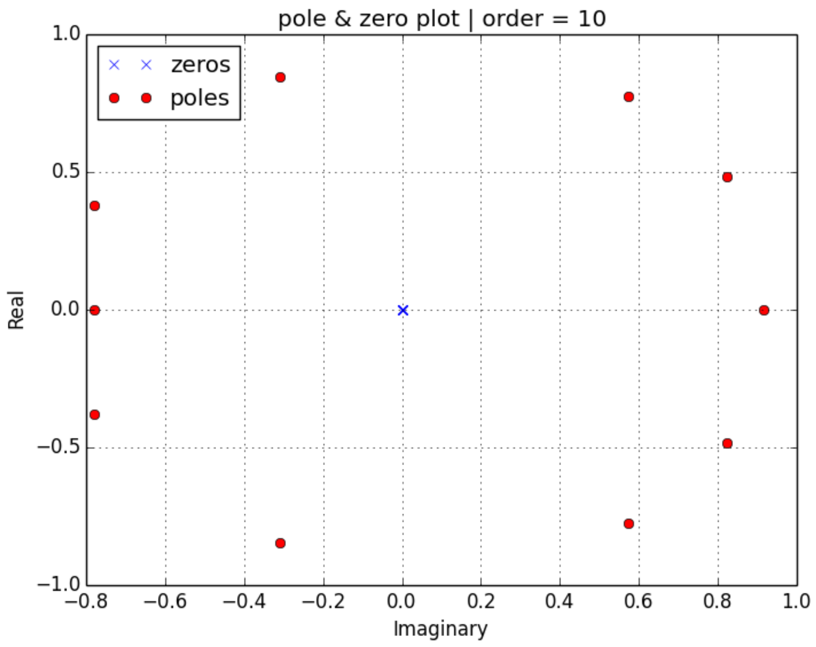

Figure 8: Poles and zeros when order = 10 for \a\from the natural recording of *machali*

```

        if order == 12 or order == 20:
            z, p, k = signal.tf2zpk(num_coeffs, coeffs)

            fig3 = plt.figure()

            plt.plot(np.real(z), np.imag(z), 'xb')
            plt.plot(np.real(p), np.imag(p), 'or')
            plt.legend(['zeros', 'poles'], loc=2)
            plt.grid()
            plt.title('pole & zero plot | order = ' + str(order) )
            plt.ylabel('real')
            plt.xlabel('imaginary')

            plt.savefig( files[file_index][-5:-4] + '_pole-zero_'+
    str(order)+'_.png')


    else:
        if order == 6 or order == 10:
            z, p, k = signal.tf2zpk(num_coeffs, coeffs)

            fig3 = plt.figure()

            plt.plot(np.real(z), np.imag(z), 'xb')
            plt.plot(np.real(p), np.imag(p), 'or')
            plt.legend(['zeros', 'poles'], loc=2)
            plt.grid()
            plt.title('pole & zero plot | order = ' + str(order) )
```

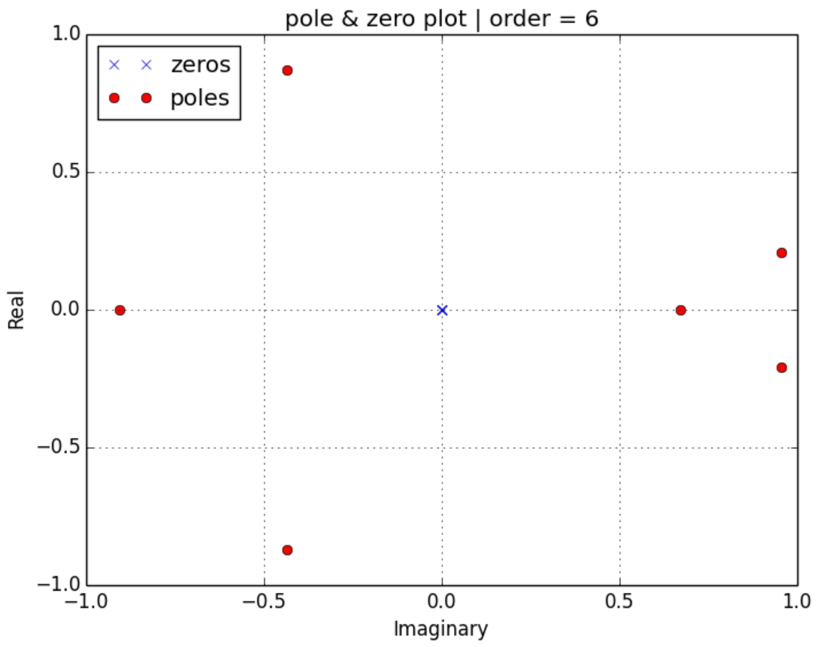

Figure 9: Poles and zeros when order = 6 for \n\from the natural recording of *machali*

```
            plt.ylabel('Real')
            plt.xlabel('Imaginary')


            plt.savefig( files[file_index][-5:-4] + '_pole-zero_'+
    str(order)+'_.png')

plt.legend(['Orig', 'Order 4', 'Order 6', 'Order 8', 'Order 10', '
    Order 12', 'Order 20'])
plt.grid()
# plt.show()
```

Listing 4: q2a.py

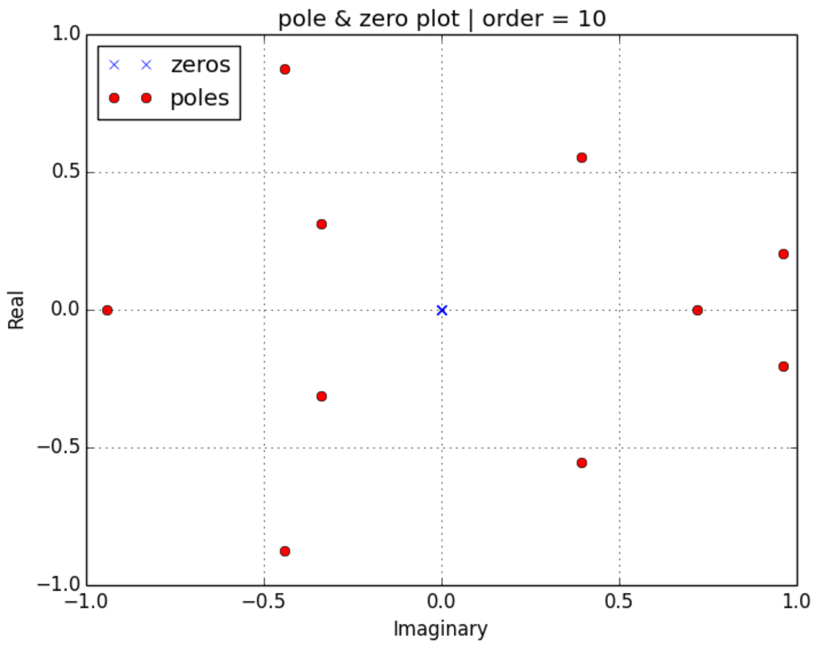

Figure 10: Poles and zeros when order = 10 for \n\from the natural recording of *machali*

Linear Predictive Coding Spectrum From the figures, we can observe that as the order increases, the LP spectrum approaches the original specrtrum. When the order is low, only the prominent peaks are taken into account and the other pecularities are handled as the order increases which makes sense since we learnt in the class that the contribution in error is more where the value of the spectrum itself is large. i.e the error in linear prediction is proportional to the absolute value of the spectrum. Hence the error at peaks will contribute more than other areas. Since we adopt least square sum minimization strategy, the peaks are modelled at lower orders.

The LPC Spectrum of \a\from the natural recording of *machali* is:

- Order 2: 15.30
- Order 4: 13.13
- Order 6: 12.84
- Order 8: 12.42
- Order 10: 10.99
- Order 12: 10.47
- Order 20: 10.07

The LPC Spectrum of \n\from the natural recording of *machali* is:

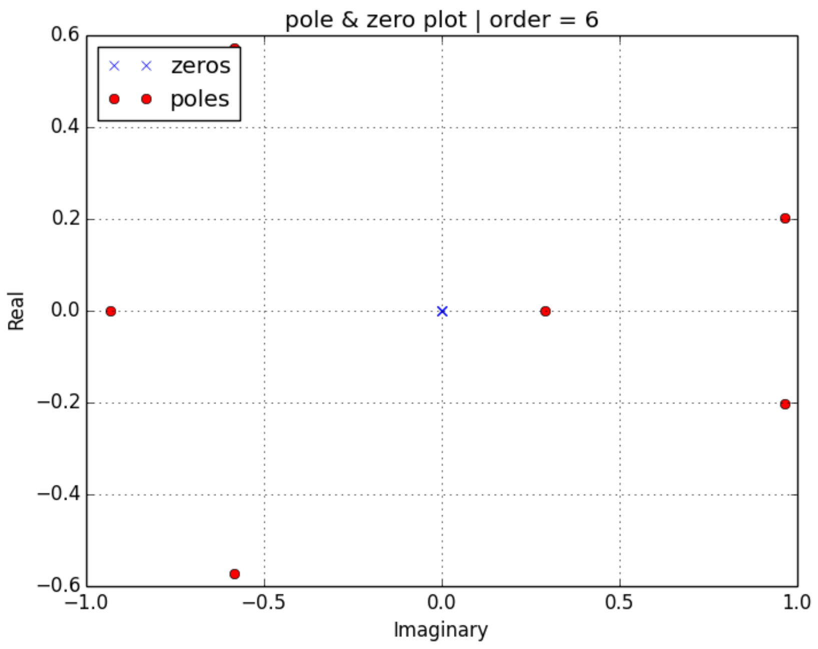

Figure 11: Poles and zeros when order = 6 for \i\from the natural recording of *machali*

- Order 2: 39.31
- Order 4: 24.83
- Order 6: 17.51
- Order 8: 17.07
- Order 10: 17.00
- Order 12: 16.58
- Order 20: 16.12

The LPC Spectrum of \I\from the natural recording of *machali* is:

- Order 2: 50.79
- Order 4: 28.53
- Order 6: 25.04
- Order 8: 22.36
- Order 10: 21.19
- Order 12: 21.05
- Order 20: 19.42

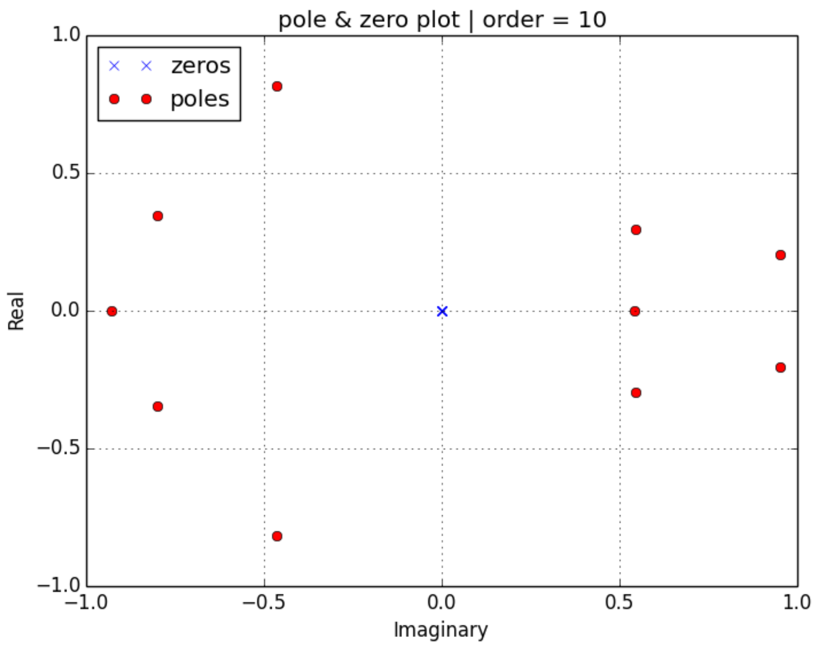

Figure 12: Poles and zeros when order = 10 for \i\from the natural recording of *machali*

The LPC Spectrum of \s\from the natural recording of *machali* is:

- Order 2: 67.89
- Order 4: 65.96
- Order 6: 45.62
- Order 8: 39.59
- Order 10: 39.07
- Order 12: 38.32
- Order 20: 36.29

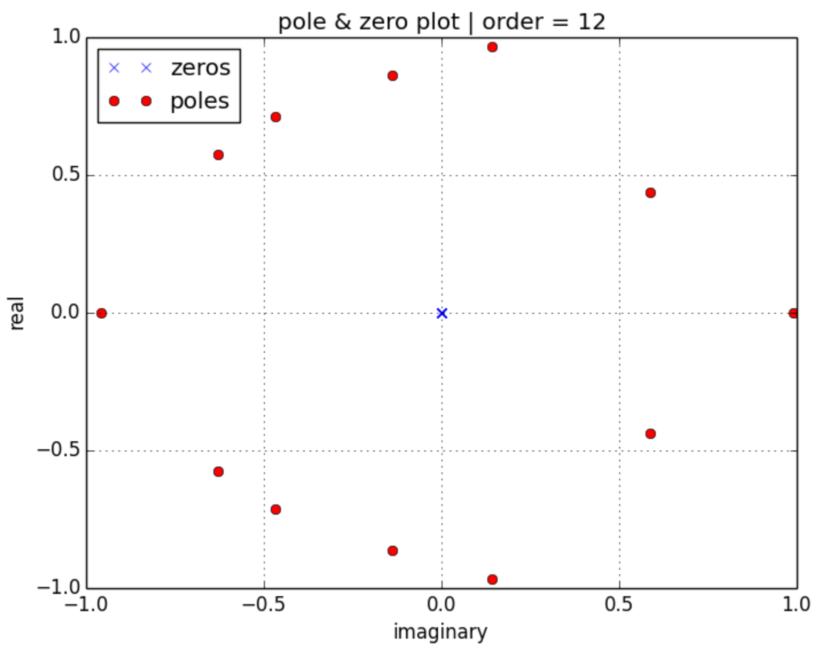

Figure 13: Poles and zeros when order = 12 for \s\from the natural recording of *machali*

Error signal energy

```
win_size = 30.0
dft_length = 1024
files = ['machali_male_8k_a.wav', 'machali_male_8k_n.wav', '
    machali_male_8k_i.wav', 'machali_male_16k_s.wav']
colors = {12: 'r', 4: 'g', 6: 'b', 8: 'y', 10: 'm', 20: 'k'}
displacements = {12: 80, 4: 0, 6: 20, 8: 40, 10: 60, 20: 100}
orders = [4, 6, 8, 10, 12, 20]
file_index = 0
```

Listing 5: hparams.py

```
import librosa
from scipy import signal
import numpy as np
from math import pi
import matplotlib.pyplot as plt
import hparams

def autocorr(x):
    result = np.correlate(x, x, mode='full')
    return result[int(result.size/2):]


files = hparams.files
file_index = hparams.file_index
colors = hparams.colors
displacements = hparams.displacements
orders = hparams.orders
```

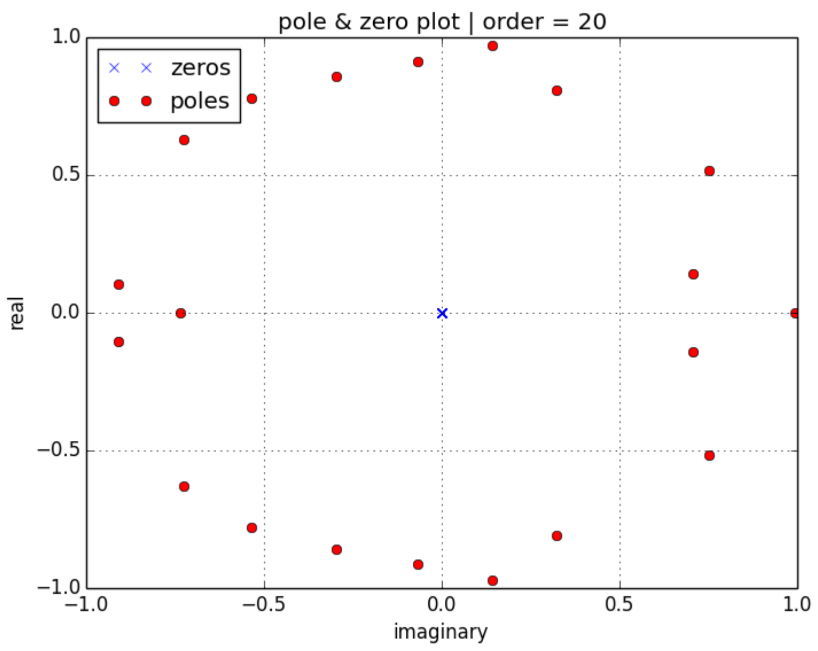

Figure 14: Poles and zeros when order = 20 for \s\from the natural recording of *machali*

```python


y, samp_freq = librosa.load(files[file_index],sr=None)
y = y/32768
win_size = hparams.win_size/1000.0
num_samples = int(samp_freq*win_size)

window = y[int((len(y)-num_samples)/2):int((len(y) + num_samples)
    /2)]*np.hamming(num_samples)

for i in range(1, len(window)):
    window[i] = window[i] - 15.0/16.0 * window[i-1]

dft = np.fft.fft(window, hparams.dft_length)
freq = np.fft.fftfreq(dft.shape[-1], 1/float(samp_freq))
plt.plot(freq[:int(len(freq)/2)], 20*np.log10(np.abs(dft[:int(len(
    dft)/2)])))

plt.title('Frequency Response of '+ files[file_index][-5:-4])
plt.ylabel('Amplitude [dB]')
plt.xlabel('Frequency')

orders = np.multiply(orders,(samp_freq/8000))
for order in orders:
    order = int(order)

    R = autocorr(window)
    error = np.zeros(order+1)
    error[0] = R[0]
```

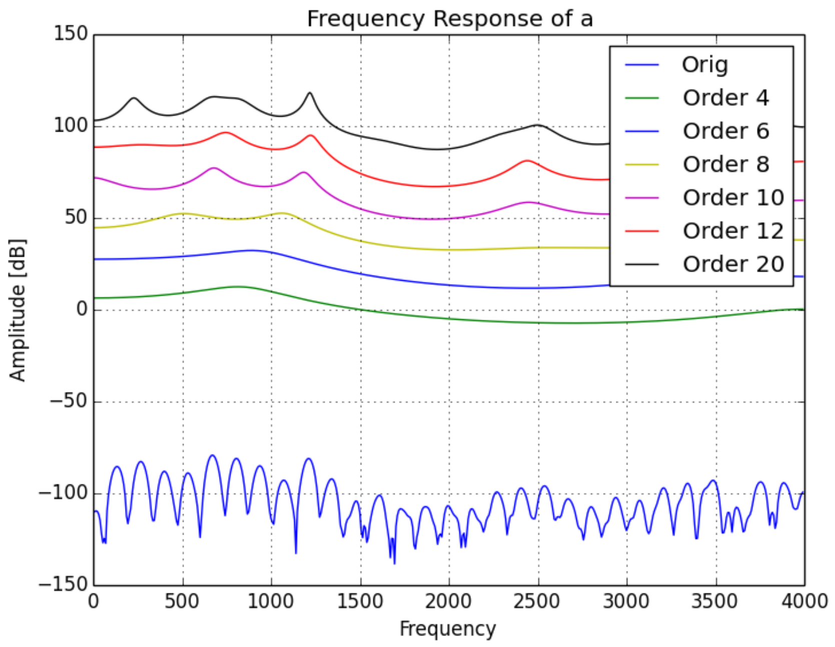

Figure 15: LPC Spectrum of \a\from the natural recording of *machali*

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

    coeffs = np.zeros(order+1)
    dummy_coeffs = np.zeros(order + 1)


    for i in range(1, order +1):
        reflec_coeffs = 0
        dummy_coeffs[1:len(coeffs)] = coeffs[1:len(coeffs)]

        for j in range(1, i):
            reflec_coeffs = reflec_coeffs + dummy_coeffs[j]*R[i-j]
        reflec_coeffs = (R[i] - reflec_coeffs)/error[i - 1]

        coeffs[i] = reflec_coeffs

        for j in range(1, i):
            coeffs[j] = dummy_coeffs[j] - reflec_coeffs*
    dummy_coeffs[i-j]

        error[i] = (1-np.square(reflec_coeffs))*error[i-1]
        G[i] = np.sqrt(error[i])
    coeffs[0] = 1.0
    coeffs[1:len(coeffs)] = -coeffs[1:len(coeffs)]
    num_coeffs = np.zeros(coeffs.shape)
    num_coeffs[0] = 1
    np.save('G_' + files[file_index][-5:-4], G)
    w, h = signal.freqz(num_coeffs, coeffs)
    plt.plot(samp_freq*w/(2*pi), displacements[int(order*8000/
    samp_freq)] + 20 * np.log10(abs(h)), colors[order*8000/
```

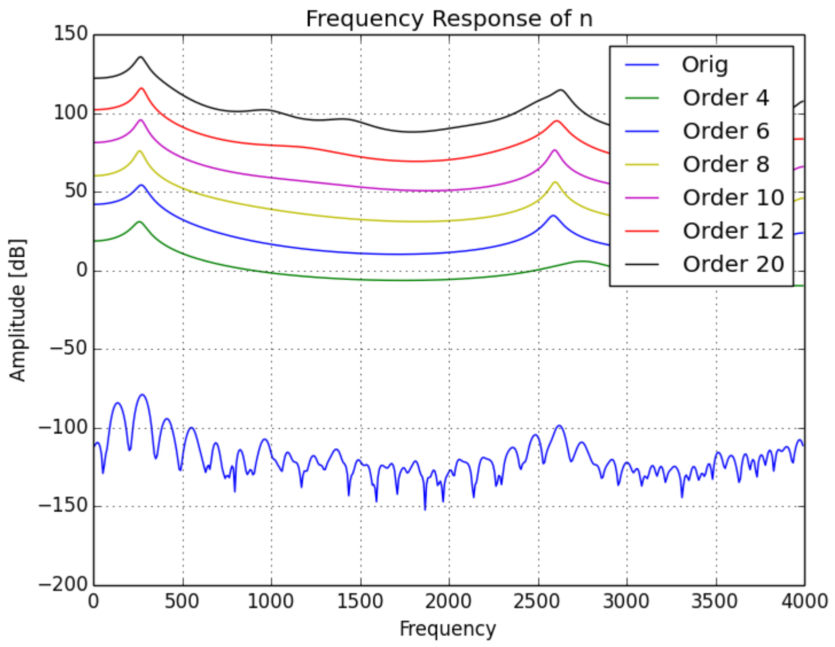

Figure 16: LPC Spectrum of \n\from the natural recording of *machali*

```
    samp_freq])

plt.legend(['Orig', 'Order 4', 'Order 6', 'Order 8', 'Order 10', '
    Order 12', 'Order 20'])
plt.grid()
plt.savefig( '../report/images/freq_resp_' + files[file_index
    ][-5:-4] + '.png')
```
Listing 6: q2b.py

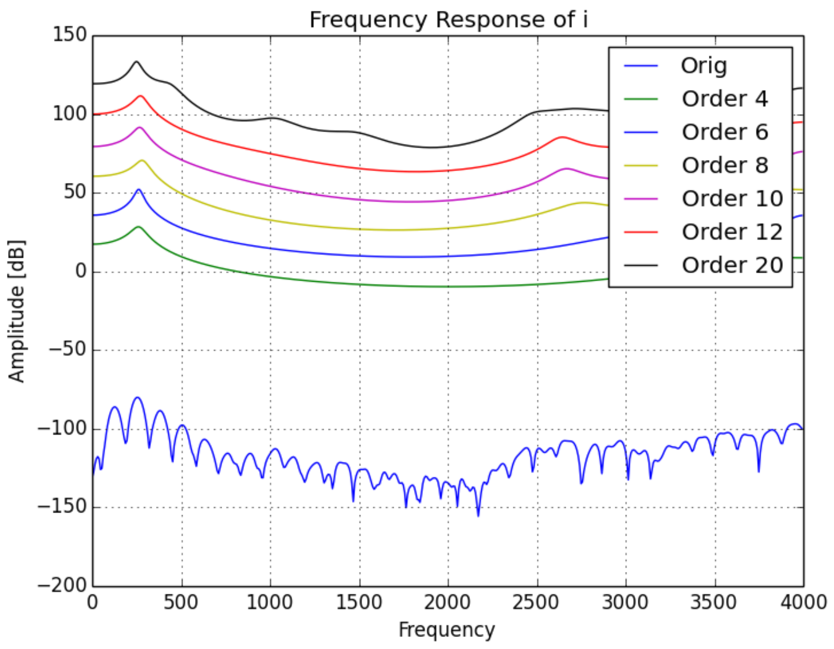

Figure 17: LPC Spectrum of \I\from the natural recording of *machali*

3 Question 3

Based on the 10th-order LPCs, carry out the inverse filtering of the \a\vowel segment and of the unvoiced sound \s\. Obtain the residual error signal in each case. Can you measure the pitch period of the voiced sound from the residual waveform? Use the acf to detect the pitch. Plot the magnitude spectrum of each of the residual signals.

3.1 Answer

From the figure, we can see that the autocorrelation function for \a\achieves a maxima (spike) at around 60 samples. Given that the sampling rate is 8 KHz, the pitch of the signal should be 8000/60 = 133.33 Hz.
We could do this since \a\is a voiced sound. On the other hand, \s\is an unvoiced sound and hence we can't observe any maxima in it't autocorrelation function. Hence we cannot calculate a pitch for this sound.

Spectrum of residual signals

```
win_size = 30.0
dft_length = 1024
files = ['machali_male_8k_a.wav', 'machali_male_8k_n.wav', '
    machali_male_8k_i.wav', 'machali_male_16k_s.wav']
file_index = 0
colors = {12: 'r', 4: 'g', 6: 'b', 8: 'y', 10: 'm', 20: 'k'}
```

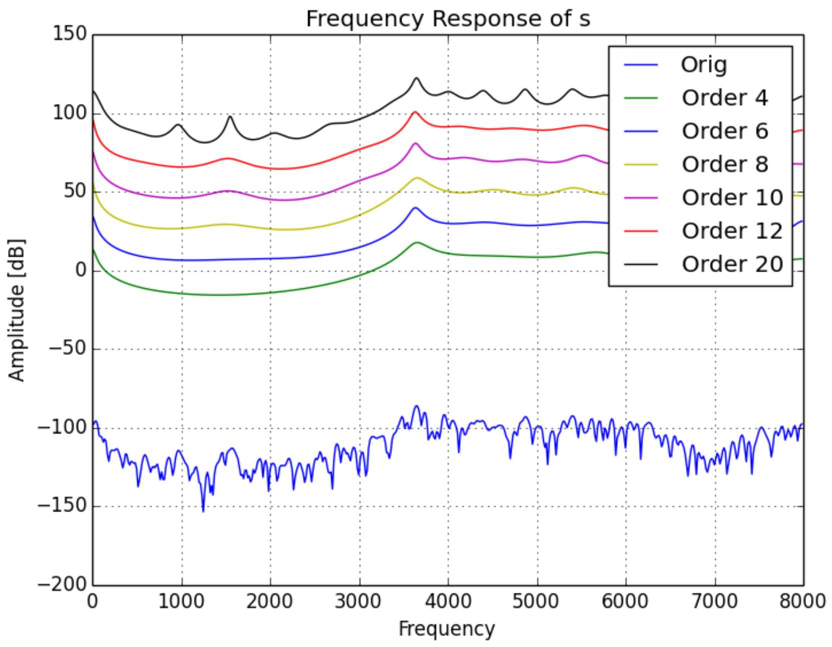

Figure 18: LPC Spectrum of \s\from the natural recording of *machali*

```
displacements = {12: 80, 4: 0, 6: 20, 8: 40, 10: 60, 20: 100}
```

Listing 7: hparams.py

```
import librosa
from scipy import signal
import numpy as np
from math import pi
import matplotlib.pyplot as plt
import hparams

def autocorr(x):
    result = np.correlate(x, x, mode='full')
    return result[int(result.size/2):]


files = hparams.files
file_index = hparams.file_index
colors = hparams.colors
displacements = hparams.displacements

y, samp_freq = librosa.load(files[file_index],sr=None)
y = y/32768
win_size = hparams.win_size/1000.0
num_samples = int(samp_freq*win_size)

window = y[int((len(y)-num_samples)/2):int((len(y)+num_samples)/2)
    ]*np.hamming(num_samples)

if file_index in [0, 1, 2]:
    for i in range(1, len(window)):
```

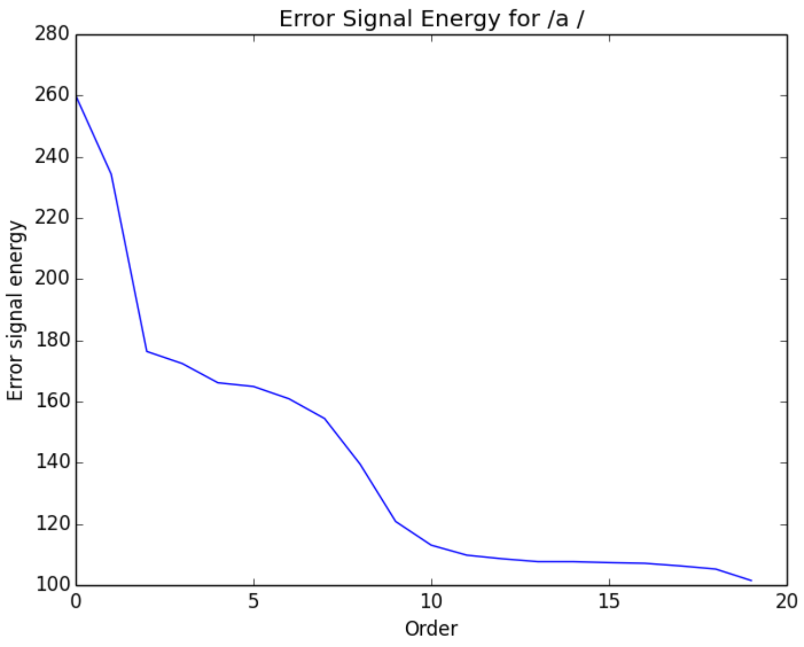

Figure 19: Error Signal Energy for \i\from the natural recording of *machali*

```
        window[i] = window[i] - 15.0/16.0 * window[i-1]


order = 10

R = autocorr(window)
error = np.zeros(order+1)
error[0] = R[0]
G = np.zeros(order + 1)

coeffs = np.zeros(order+1)
dummy_coeffs = np.zeros(order + 1)


for i in range(1, order +1):
    reflec_coeffs = 0
    dummy_coeffs[1:len(coeffs)] = coeffs[1:len(coeffs)]

    for j in range(1, i):
        reflec_coeffs = reflec_coeffs + dummy_coeffs[j]*R[i-j]
    reflec_coeffs = (R[i] - reflec_coeffs)/error[i - 1]

    coeffs[i] = reflec_coeffs

    for j in range(1, i):
        coeffs[j] = dummy_coeffs[j] - reflec_coeffs*dummy_coeffs[i-
    j]

    error[i] = (1-np.square(reflec_coeffs))*error[i-1]
    G[i] = np.sqrt(error[i])
```

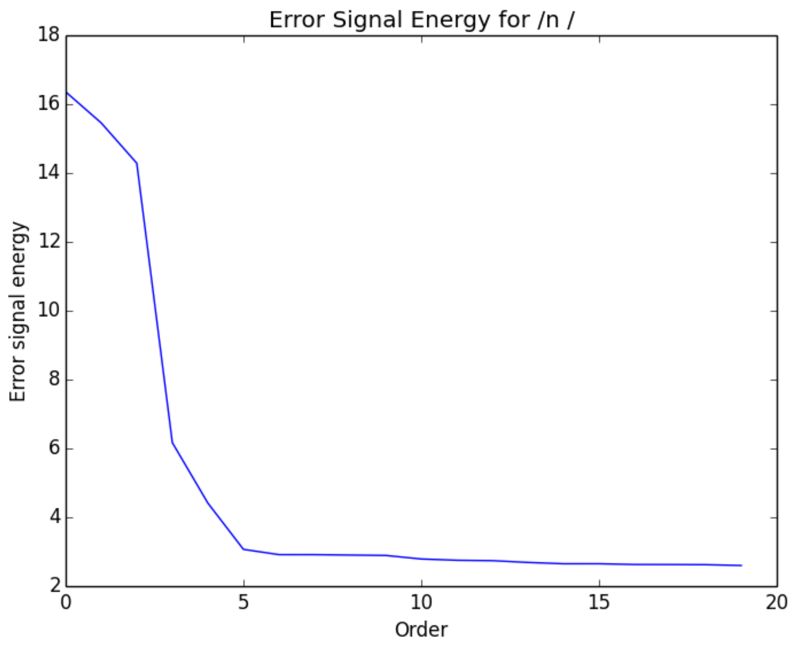

Figure 20: Error Signal Energy for \i\from the natural recording of *machali*

```

coeffs[0] = 1.0
coeffs[1:len(coeffs)] = -coeffs[1:len(coeffs)]
num_coeffs = np.zeros(coeffs.shape)
num_coeffs[0] = 1

# window_filter = signal.lfilter(coeffs, num_coeffs, window)
# fig1 = plt.figure()
# dft = np.fft.fft(window_filter, 1024)
# freq = np.fft.fftfreq(dft.shape[-1], 1/float(samp_freq))
# plt.title('Residual Signal for ' + files[file_index][-5:-4])
# plt.ylabel('DFT Amplitude', color='b')
# plt.xlabel('Frequency')
# plt.grid()
# plt.plot(freq[:len(freq)/2], 20*np.log10(np.abs(dft[:len(dft)/2])
    ), 'b')

# fig2 = plt.figure()
# plt.title('Residual autocorrelation for ' + files[file_index
    ][-5:-4])
# plt.ylabel('Amplitude', color='b')
# plt.xlabel('Sample')
# plt.grid()
# plt.plot(autocorr(window_filter))

w, h = signal.freqz(num_coeffs, coeffs)

if file_index == 3:
    order = int((order*samp_freq/8000))
    plt.plot(samp_freq*w/(2*pi), displacements[order*8000/samp_freq
```

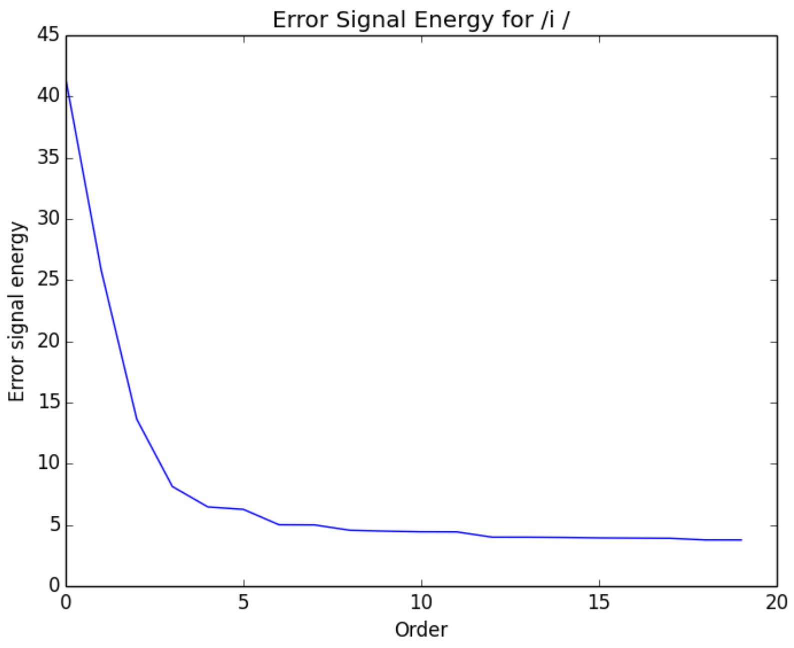

Figure 21: Error Signal Energy for \s\from the natural recording of *machali*

```
    ] + 20 * np.log10(abs(h)), colors[order*8000/samp_freq])
else:
    plt.plot(samp_freq*w/(2*pi), displacements[order*8000/samp_freq
    ] + 20 * np.log10(abs(h)), colors[order*8000/samp_freq])
plt.legend(['Order 10'])
plt.grid()

plt.show()
```

Listing 8: q3.py

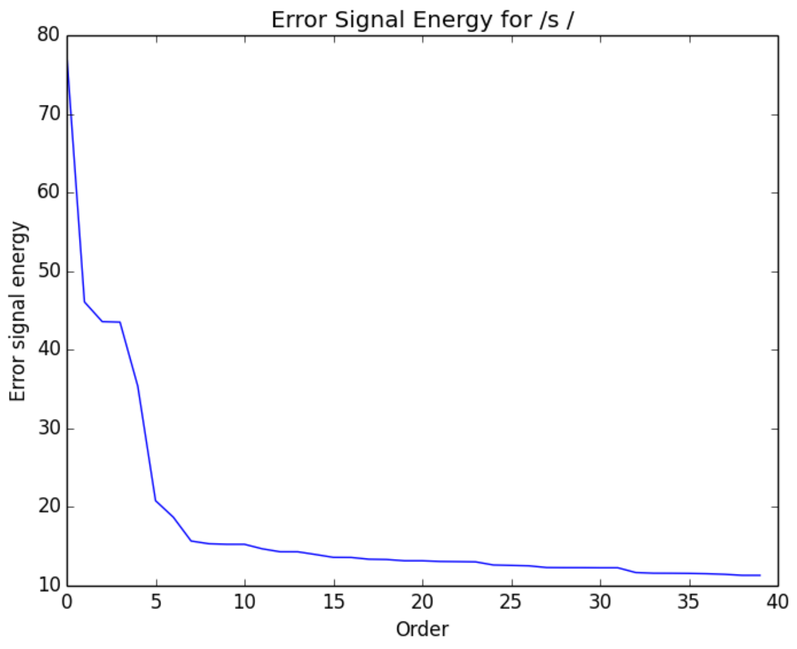

Figure 22: Error Signal Energy for \s\from the natural recording of *machali*

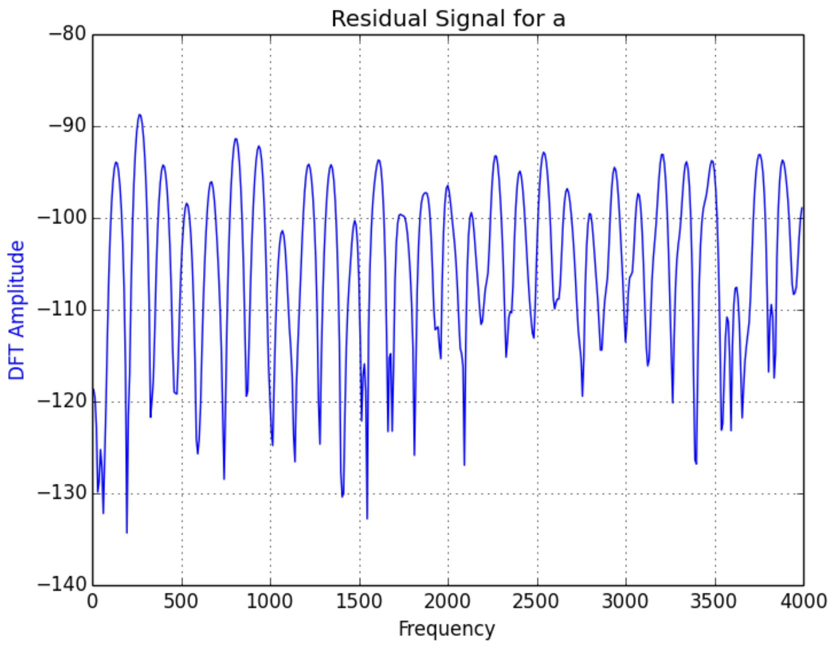

Figure 23: Residual signal for \a\from the natural recording of *machali*

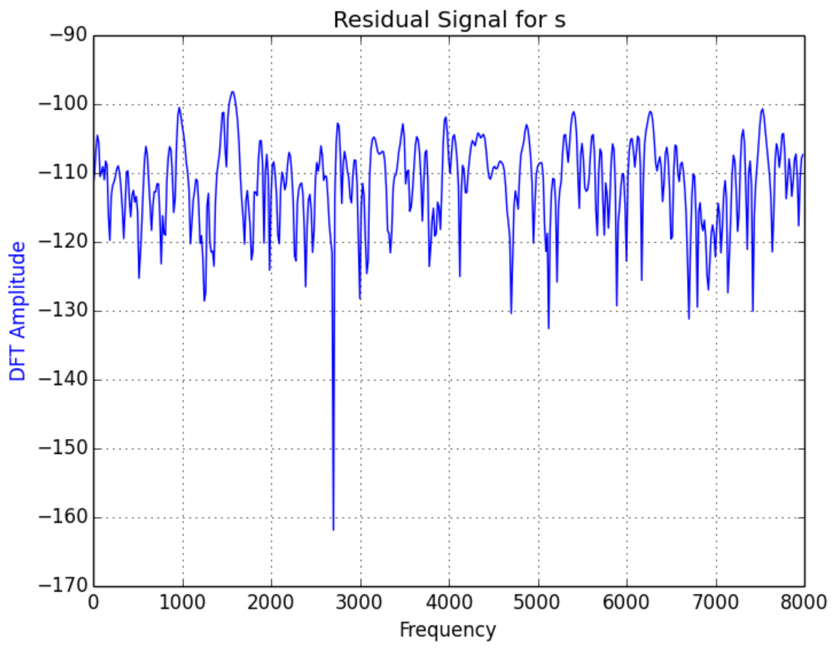

Figure 24: Residual signal for \s\from the natural recording of *machali*

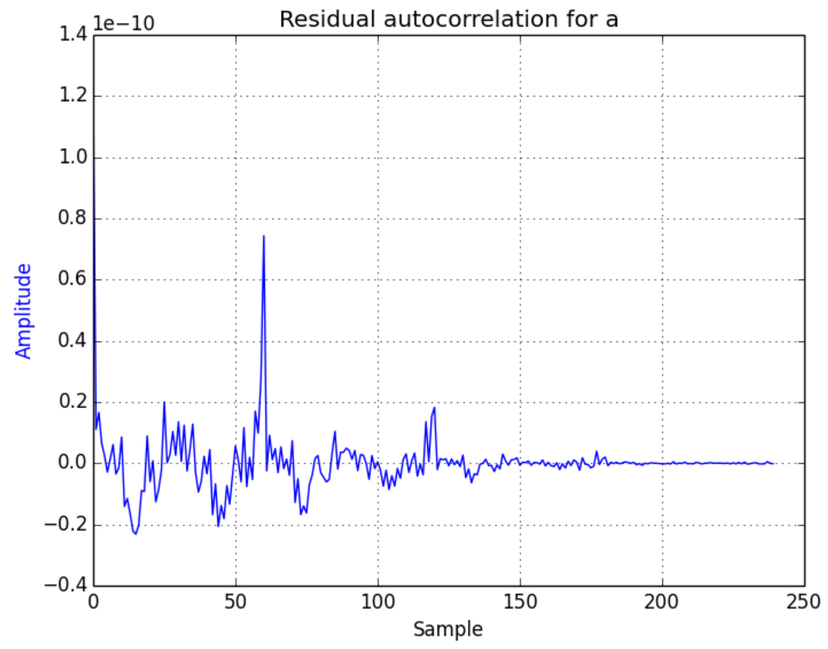

Figure 25: Autocorrelation of the residual signal for \a\from the natural recording of *machali*

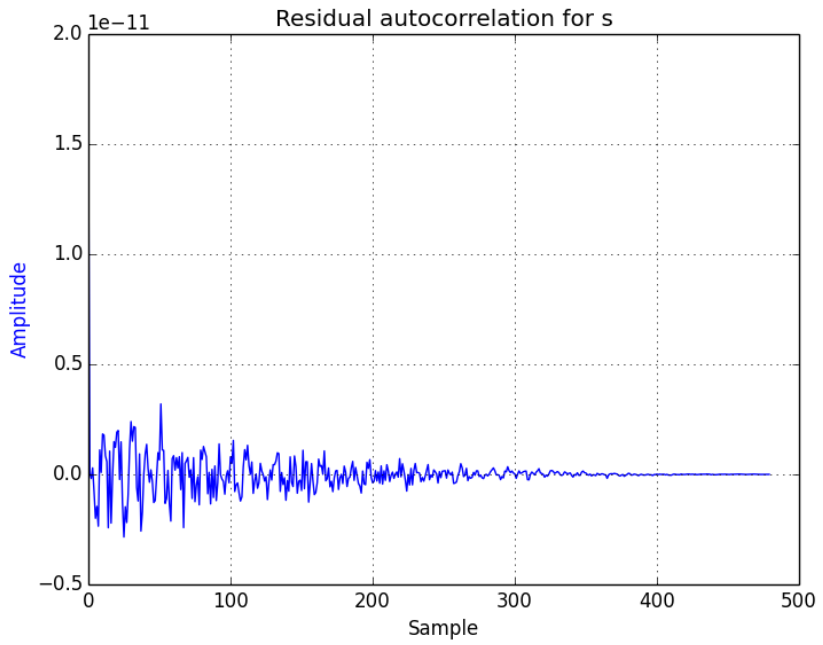

Figure 26: Autocorrelation of the residual signal for \s\from the natural recording of *machali*

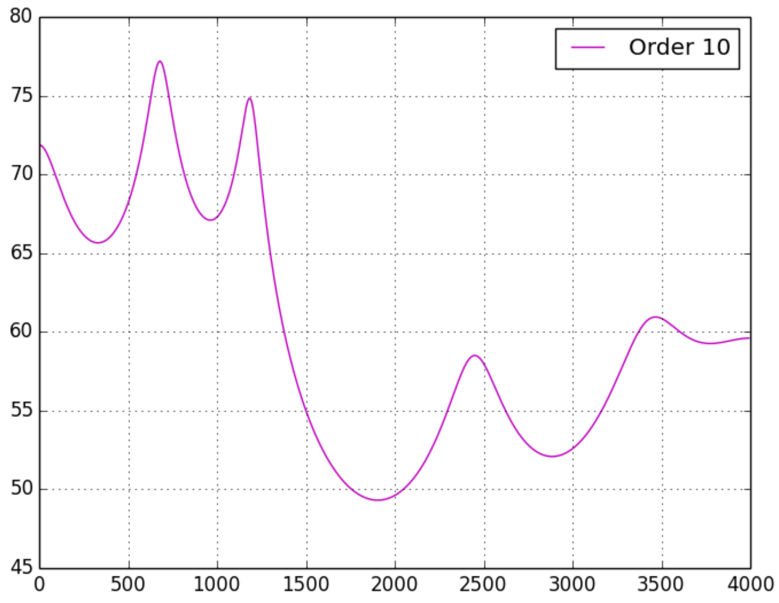

Figure 27: Spectrum of the residual signal for \a\from the natural recording of *machali*

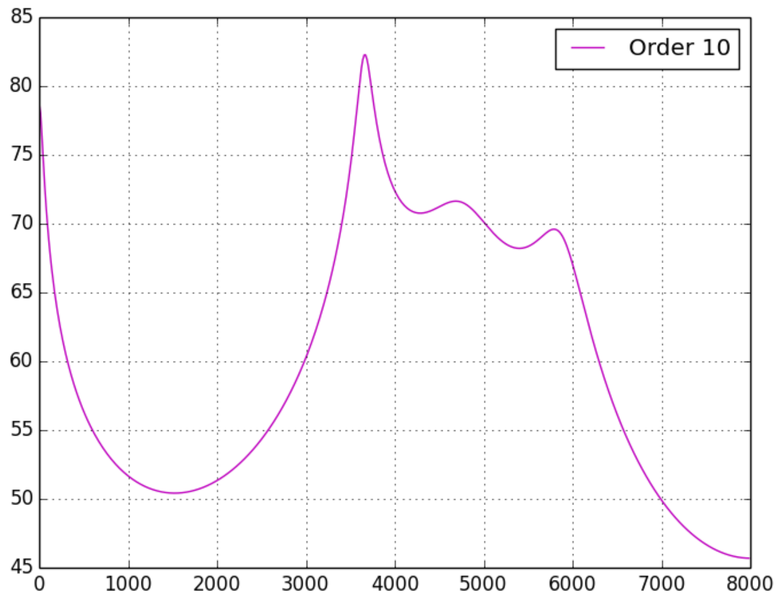

Figure 28: Spectrum of the residual signal for \s\from the natural recording of *machali*