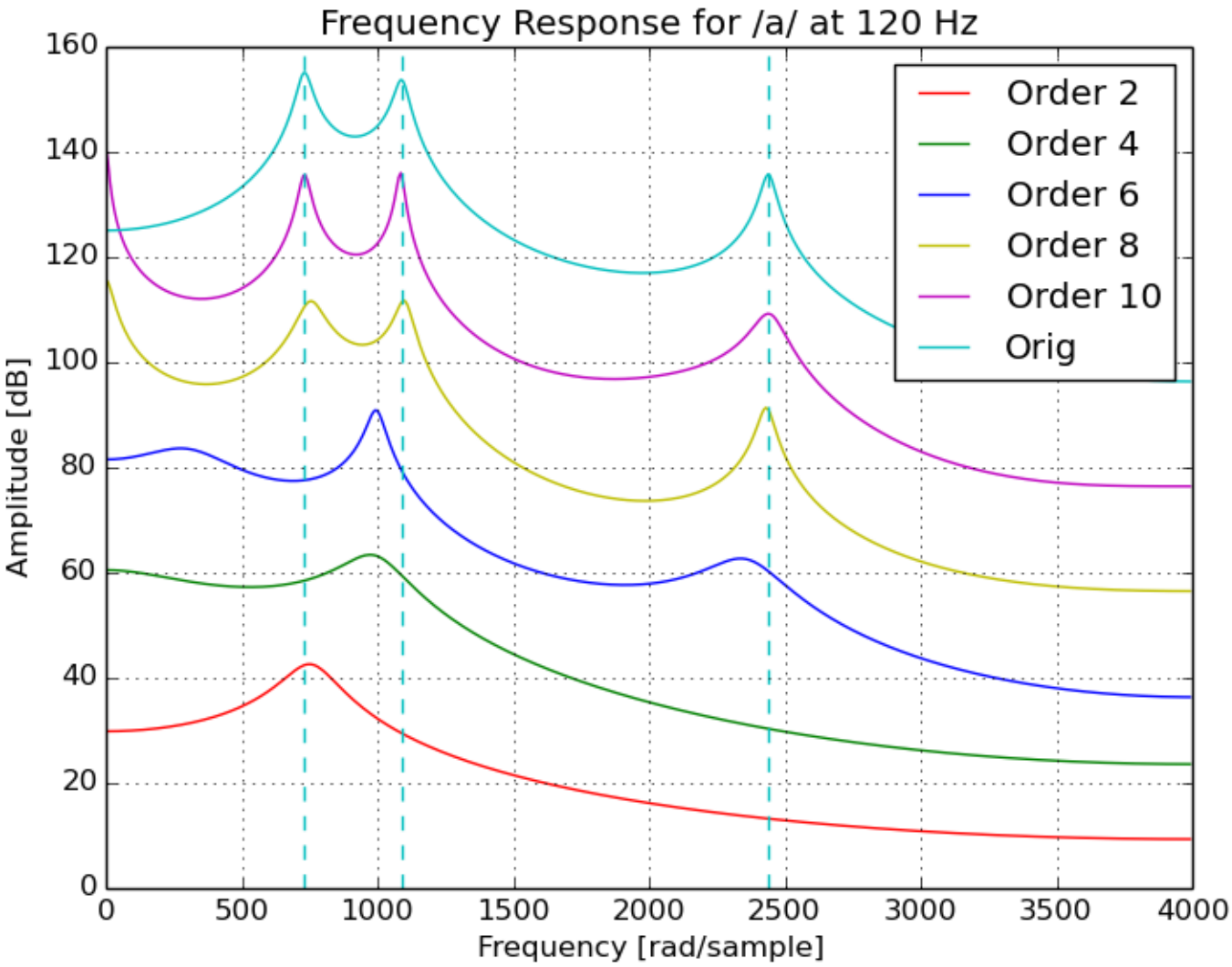
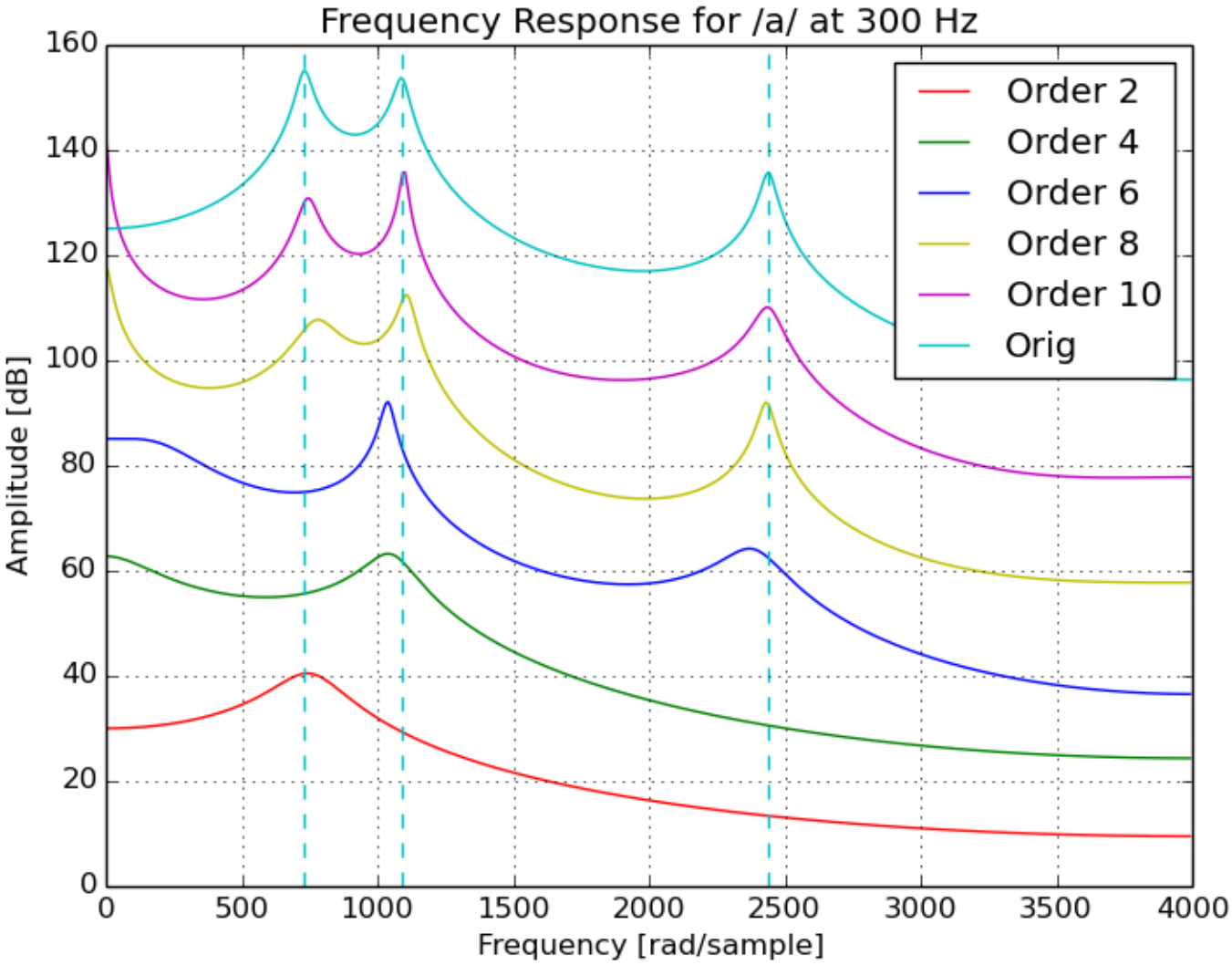
EE679: Computing Assignment 2

Swapnil Bembde 14D070034

October 1, 2017

A. Synthesized vowel:





comments:

* As we increase the order(p), frequency response looks similar to the original spectral envelope. Hence, we can infer that estimated LPC spectrum approximates the real spectrum better as we increase LPC order.
* Also, we can observe, as p increases the number of peaks increases. Number of peaks is half of the order. This can also be related in another fashion, i.e. as order increase number of poles increase, and peaks are at the formants.
* There is not much difference in both plots; but there are small differences. As we increase pitch of the vowel LPC estimated spectrum approximates well.

Code snippet for part A:

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

win_in = data[:Samp]*np.hamming(Samp)    #Samp = samplingfreq*windowTime
# Use Levinson's algorithm to calculate a[i]
for filterOrder in [2, 4, 6, 8, 10]:
    r = autocorr(win_inp) #autocorrelation of windowed input signal
    a = np.zeros(filterOrder+1)
    ao = np.zeros(a.shape)
    E = r[0]
```

```
    for i in range(1, filterOrder+1):
        k = 0
        Eo = E
        ao[1:len(a)] = a[1:len(a)]
        # finding K
        for l in range(1, i):
            k = k + ao[l]*r[i-l]
        k = (r[i] - k)/Eo

        # calculation of a's
        a[i] = k
        for l in range(1, i):
            a[l] = ao[l] - k*ao[i-l]
        E = (1-k*k)*Eo

    a[0] = 1.0
    a[1:len(a)] = -a[1:len(a)]
    b = np.zeros(a.shape)
    b[0] = 1
    #trasfer function
    w, h = signal.freqz(b, a)

# calculation of orignal signal
formants = np.asarray([730, 1090, 2440])
bandwidths = np.asarray([50, 50, 50])

# Calculate pole angles and radii
r = np.exp(-pi*bandwidths/fSamp)
theta = 2*pi*formants/fSamp

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

# Getting transfer function
b, a = signal.zpk2tf(zeros, poles, 1)

# Getting frequency response
wf, hf = signal.freqz(b, a)
```

B. Natural speech:

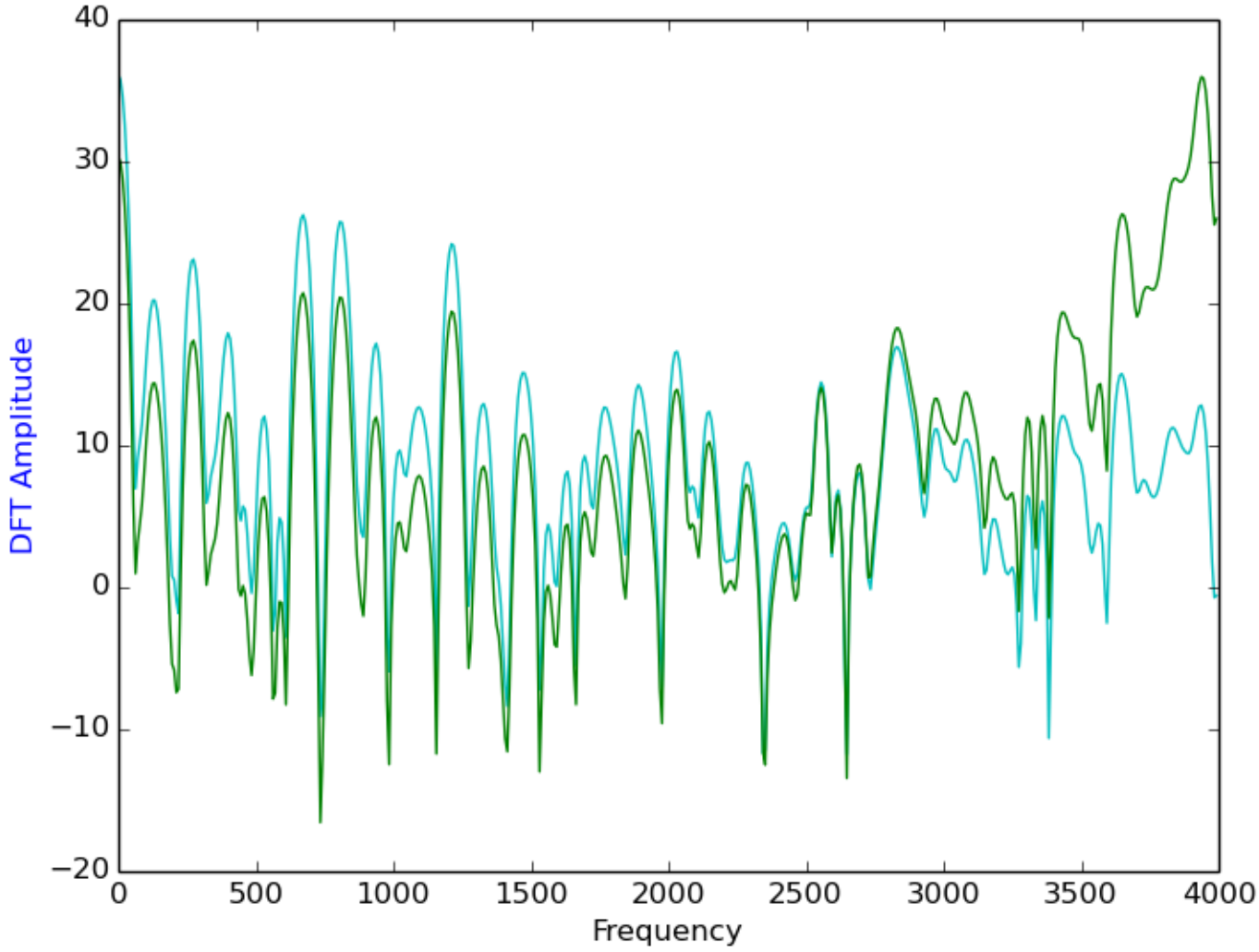
1 : Pre-emphasis:

Original narrowband spectrum is in 'Cyan' and pre-emphasized spectrum is in 'Green'. Assumed high pass filter with 0.95 as it's zero.

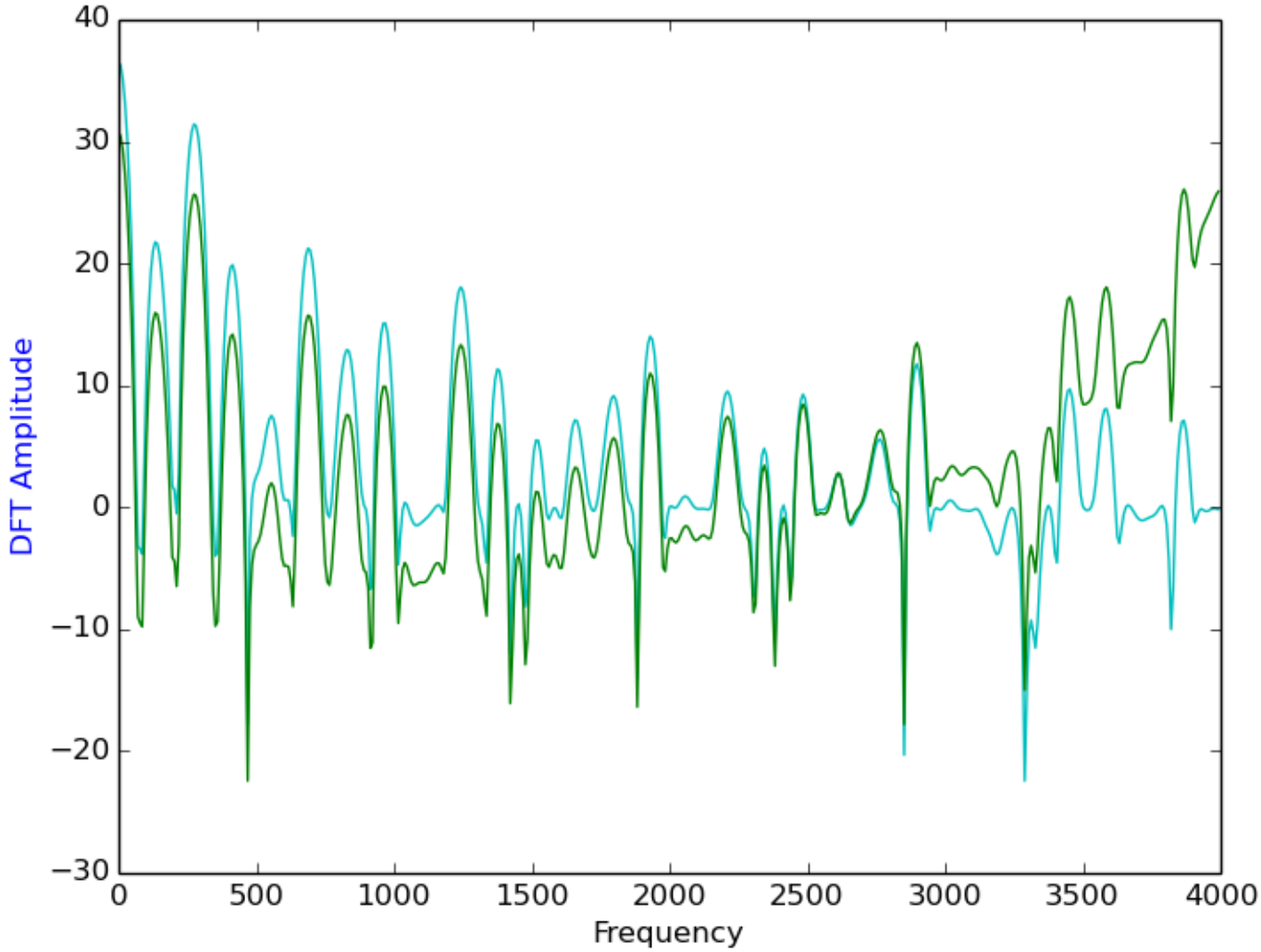
* As we can clearly see, higher frequency components' amplitude has increased after pre-emphasis. This decreases glottal roll off of the speech.

* Pre-emphasis will not much effect on /s/. And, pre-emphasis for /s/ is not done because this is model involves assumptions and /s/ is unvoiced fricative.

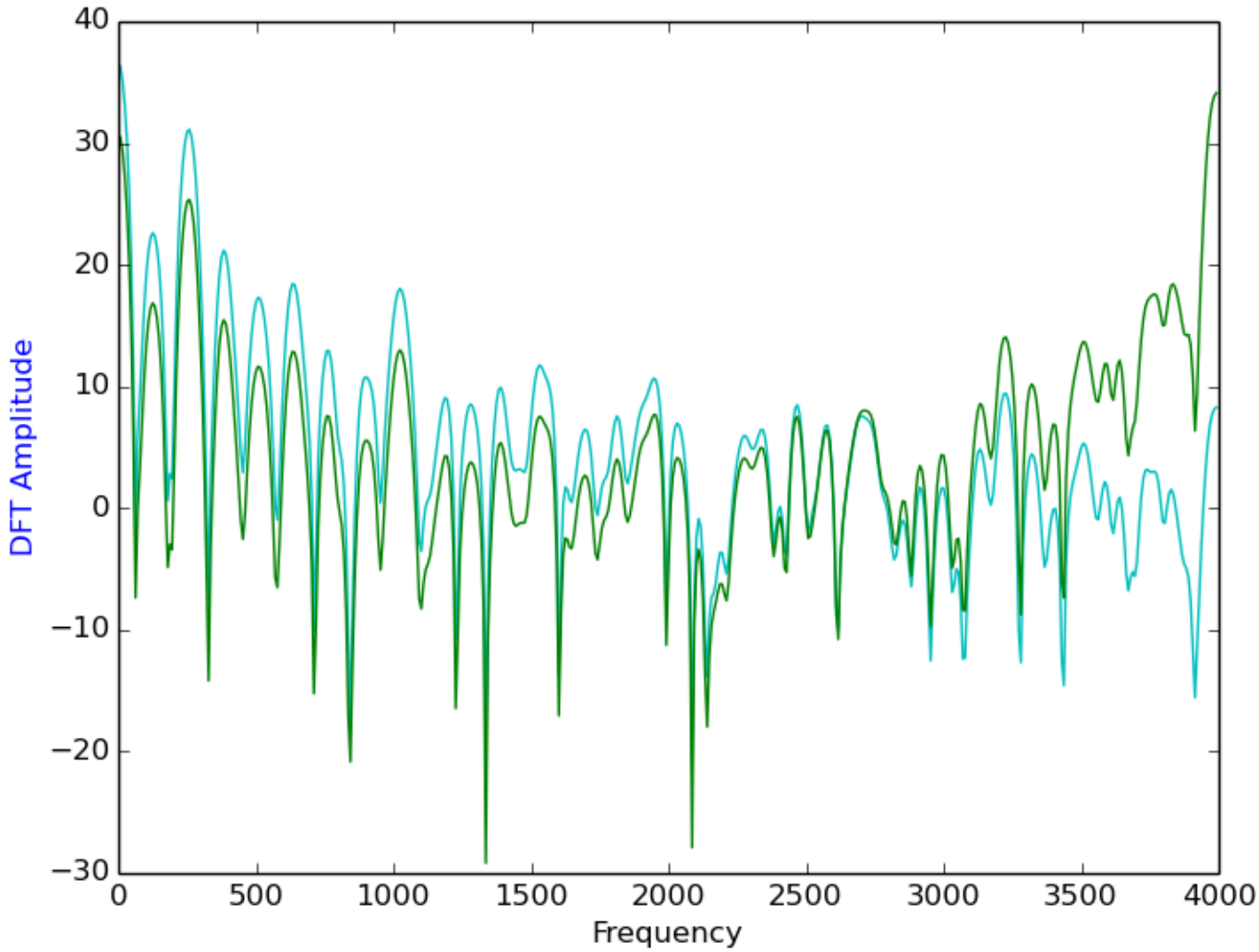
Spectrum for /a/



Spectrum for /n/



Spectrum for /i/



Code snippet for pre-emphasis part :

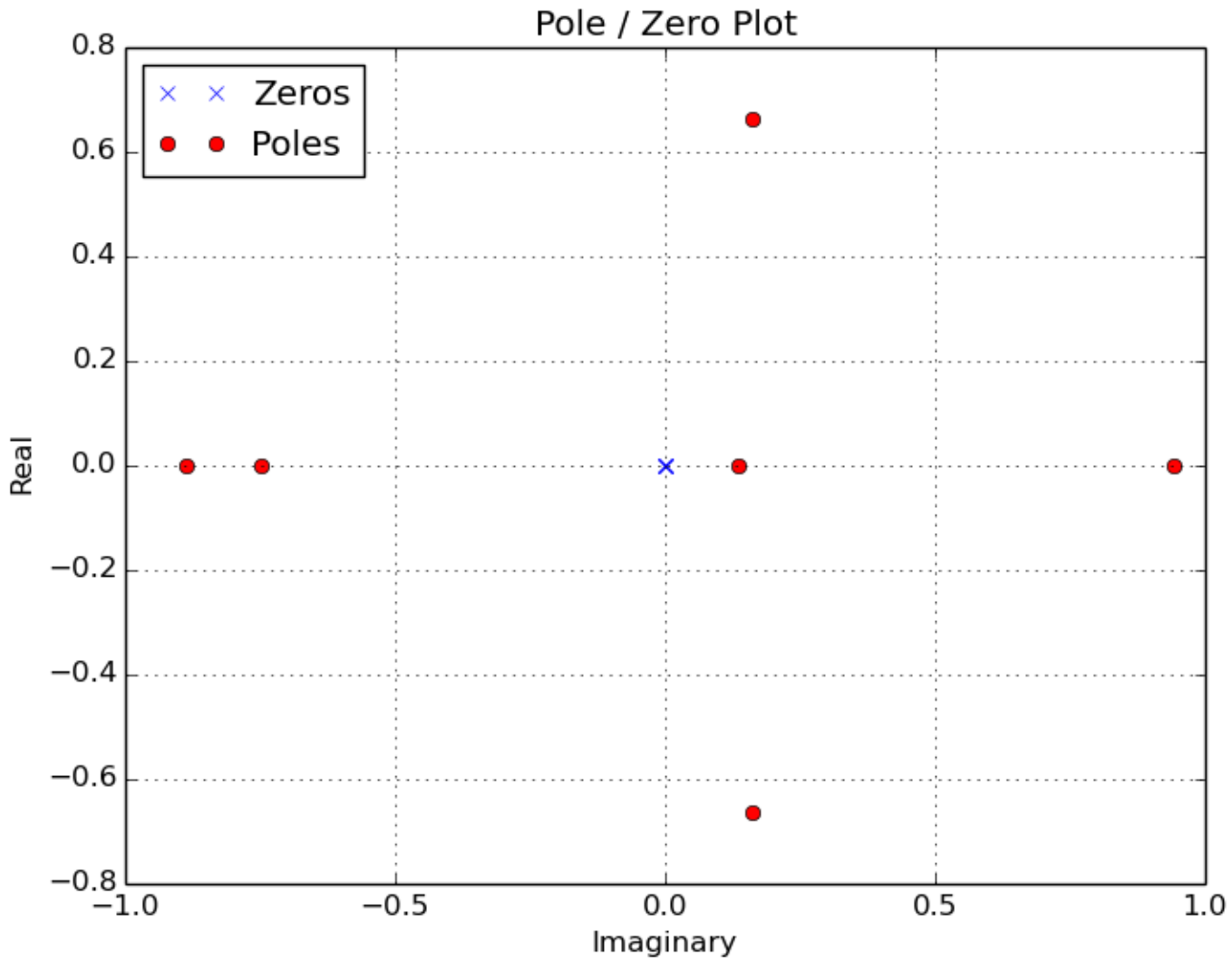
```
#read the data
[fSamp, data] = read(files[sys.argv[1]])

winT = 30.0/1000.0 #window length
Samples = fSamp*winT
#take centered segment
w_in = data[int((len(data)-Samples)/2):int((len(data)+Samples)/2)]*np.hamming(Samples)

# Plot the spectrum of this windowed signal
dft1 = np.fft.fft(win, 1024)
freq1 = np.fft.fftfreq(dft1.shape[-1], 1/float(fSamp))

#pre-emphasising the signal
for i in range(1, len(win)):
    w_in[i] = w_in[i] - 0.95 * win[i-1]

dft2 = np.fft.fft(w_in, 1024)
freq2 = np.fft.fftfreq(dft2.shape[-1], 1/float(fSamp))
```

Pole - Zero plots

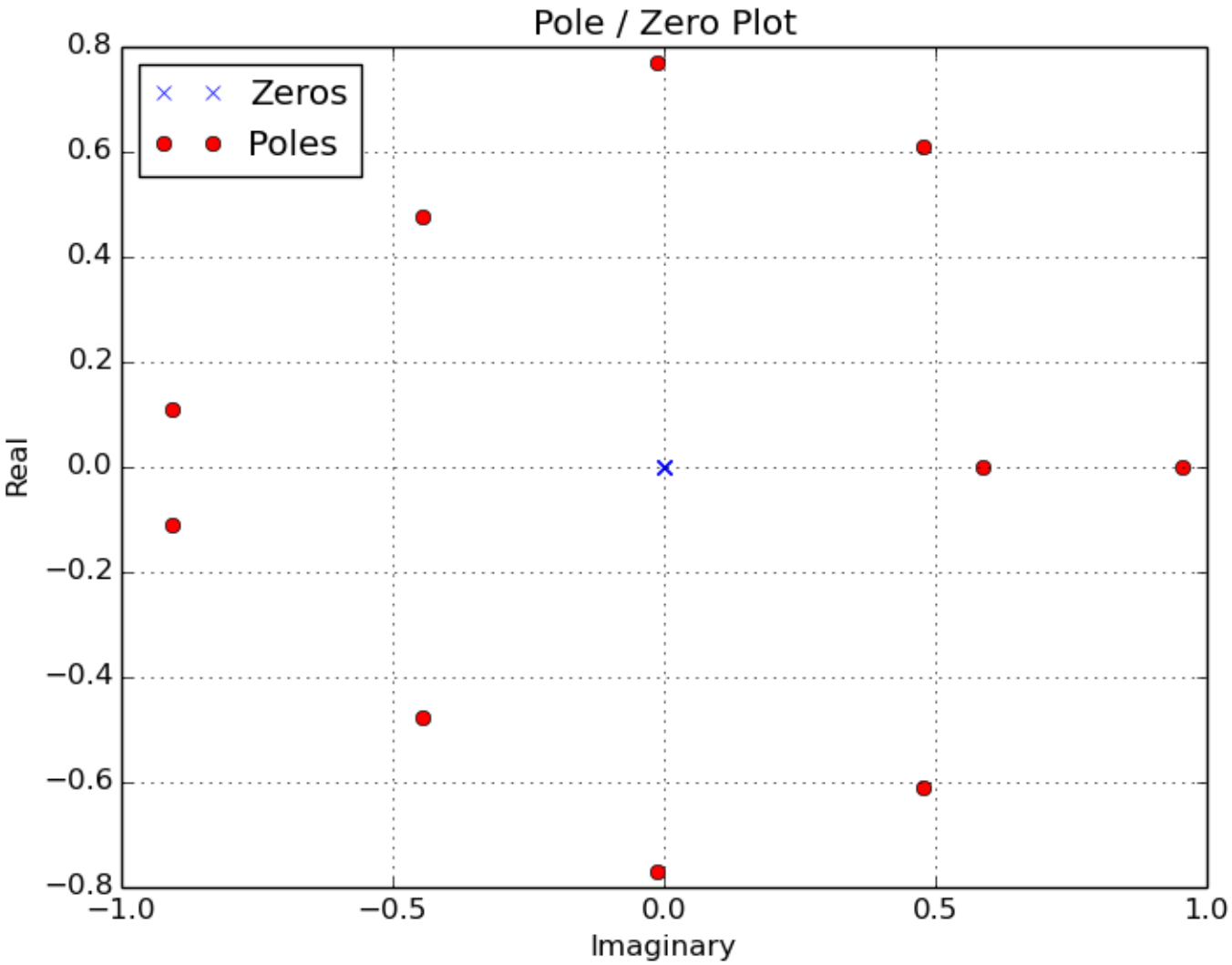
Using autocorrelation coefficients and gain following plots have been plotted.

For /a/

p = 6

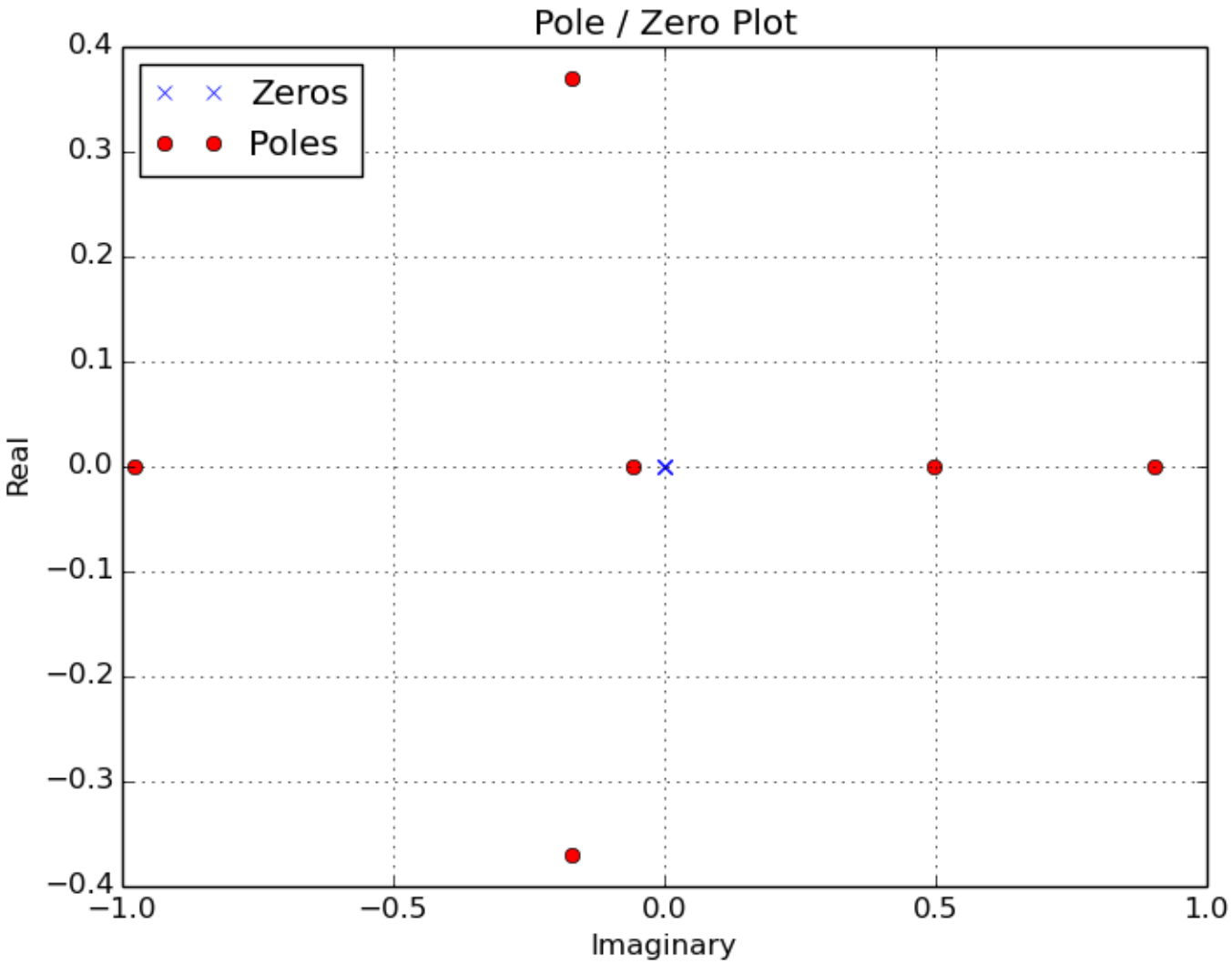


p = 10

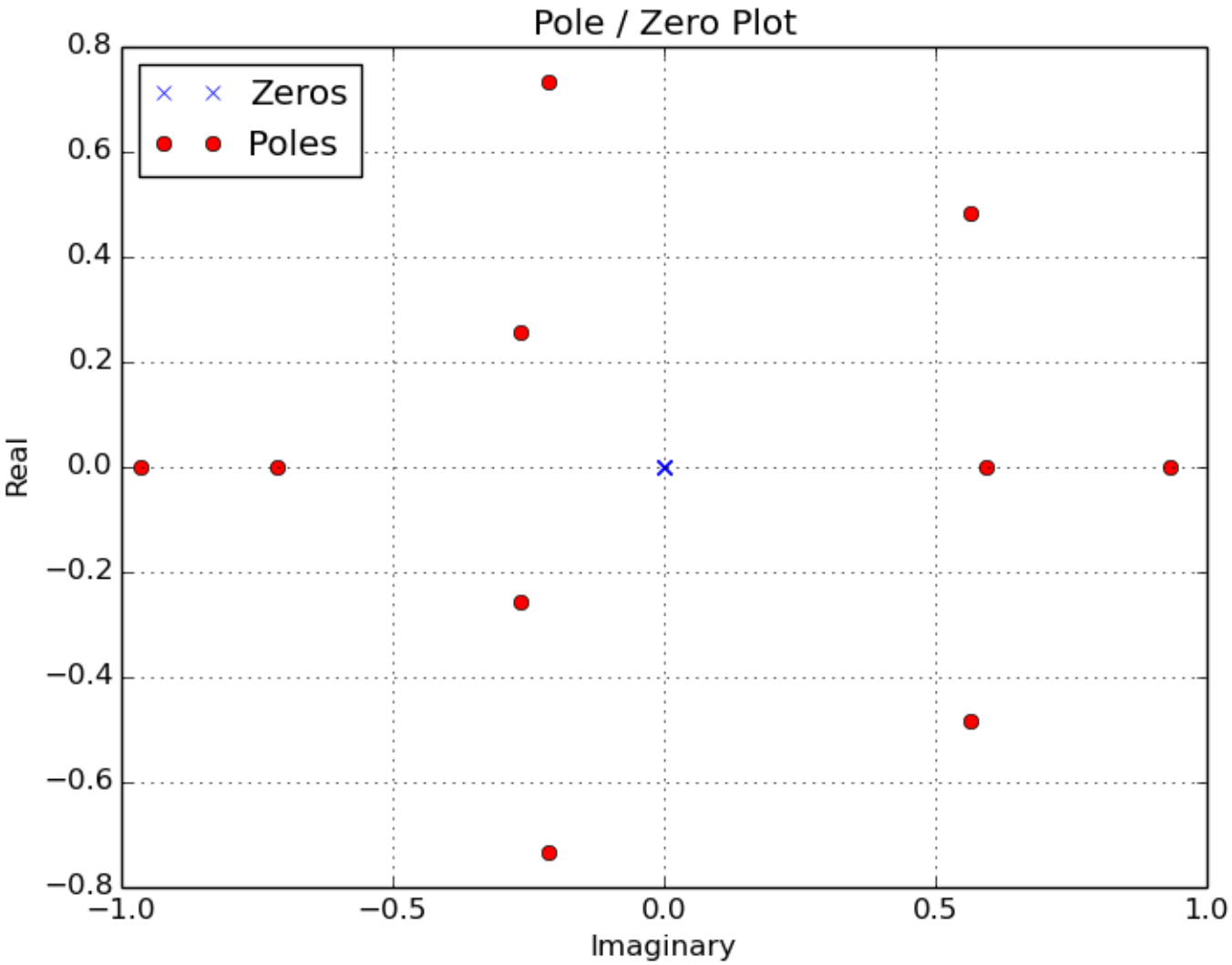

Pole / Zero Plot
Zeros
Poles
Real
Imaginary
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−0.5
0.0
0.5
1.0

For /n/

p = 6

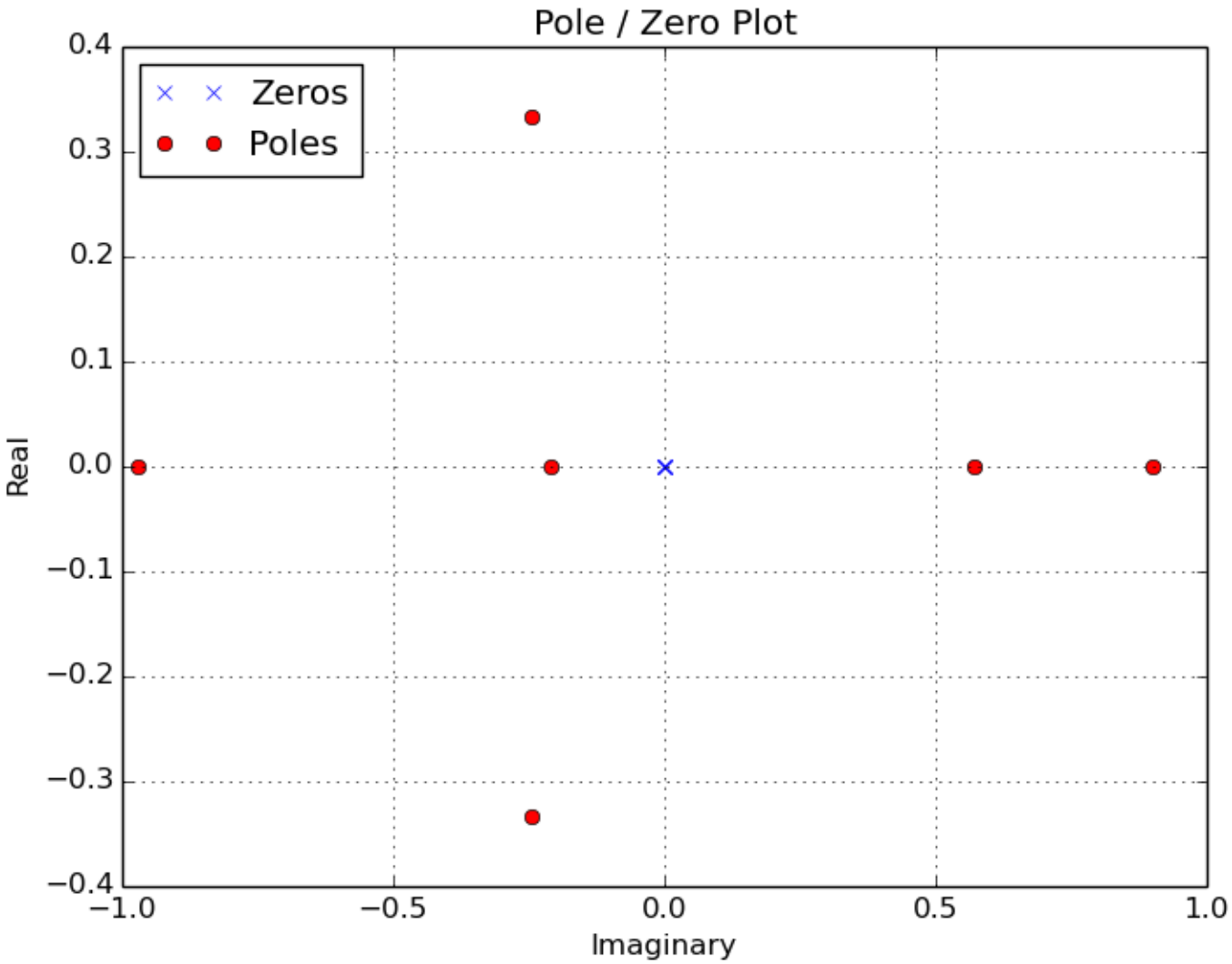

Pole / Zero Plot
Zeros
Poles
Real
Imaginary
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−1.0
−0.5
0.0
0.5
1.0

p = 10

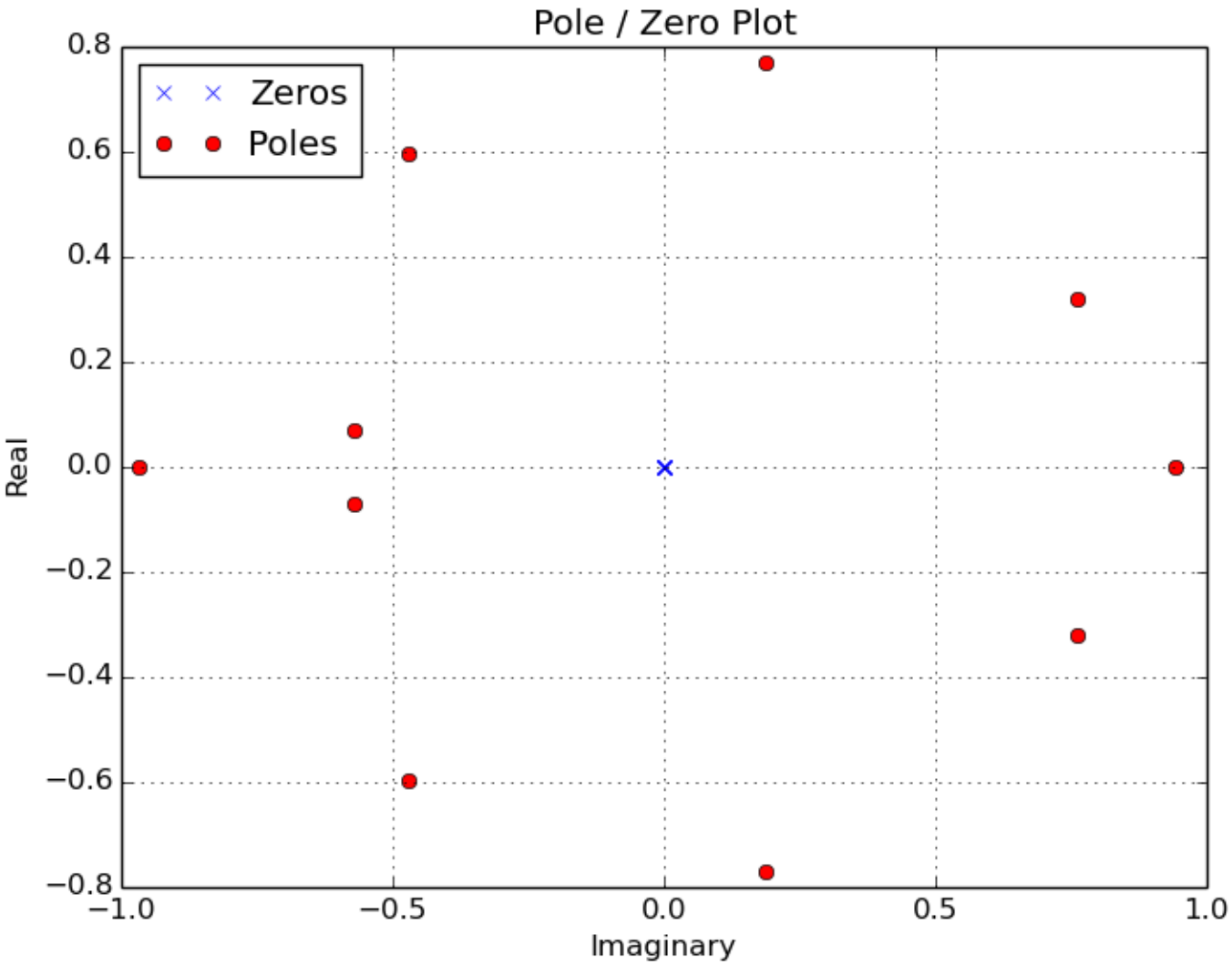

Pole / Zero Plot
Zeros
Poles
Real
Imaginary
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−0.5
0.0
0.5
1.0

For /i/

p = 6

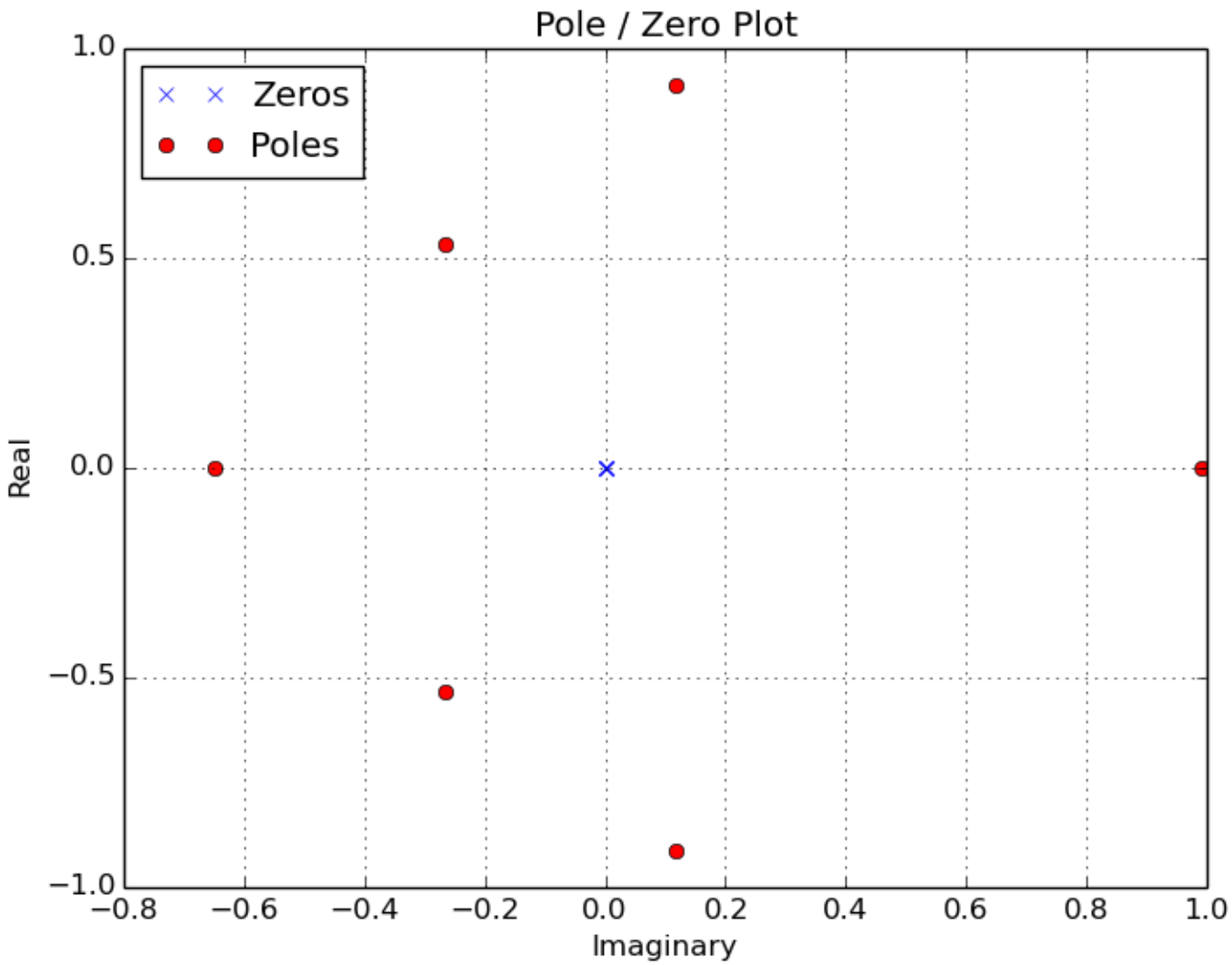

Pole / Zero Plot
Zeros
Poles
Real
Imaginary
0.4
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
−0.4
−1.0
−0.5
0.0
0.5
1.0

p = 10

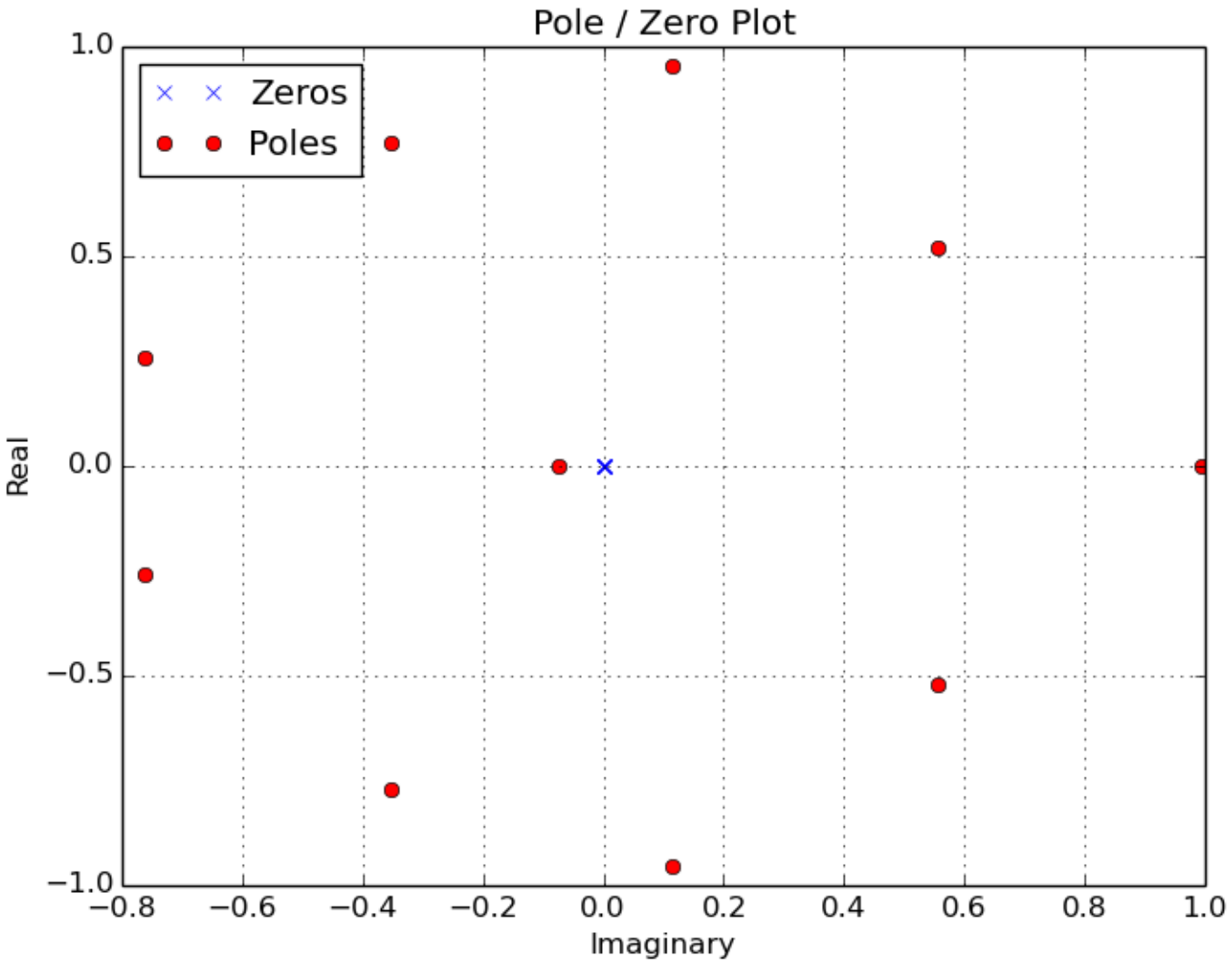

Pole / Zero Plot
Zeros
Poles
Real
Imaginary
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−0.5
0.0
0.5
1.0

For /s/

p = 6


Pole / Zero Plot
Zeros
Poles
1.0
0.5
0.0
−0.5
−1.0
Real
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
Imaginary

p = 10



Comments:

*This plots completely depend on the speech signal. Using LD algorithm, different input signal will cause different pole-zero plots.

*For /s/, it's taken from 16Khz sampled signal. Hence it has frequency components upto 8k. So for better approximation, we should have more order than other segments. We should double the order so that p=6 and p=10 will correspond to p=12 and p=20. But from the question it was not clear, so I have included p=6 and p=10.

Code snippet:

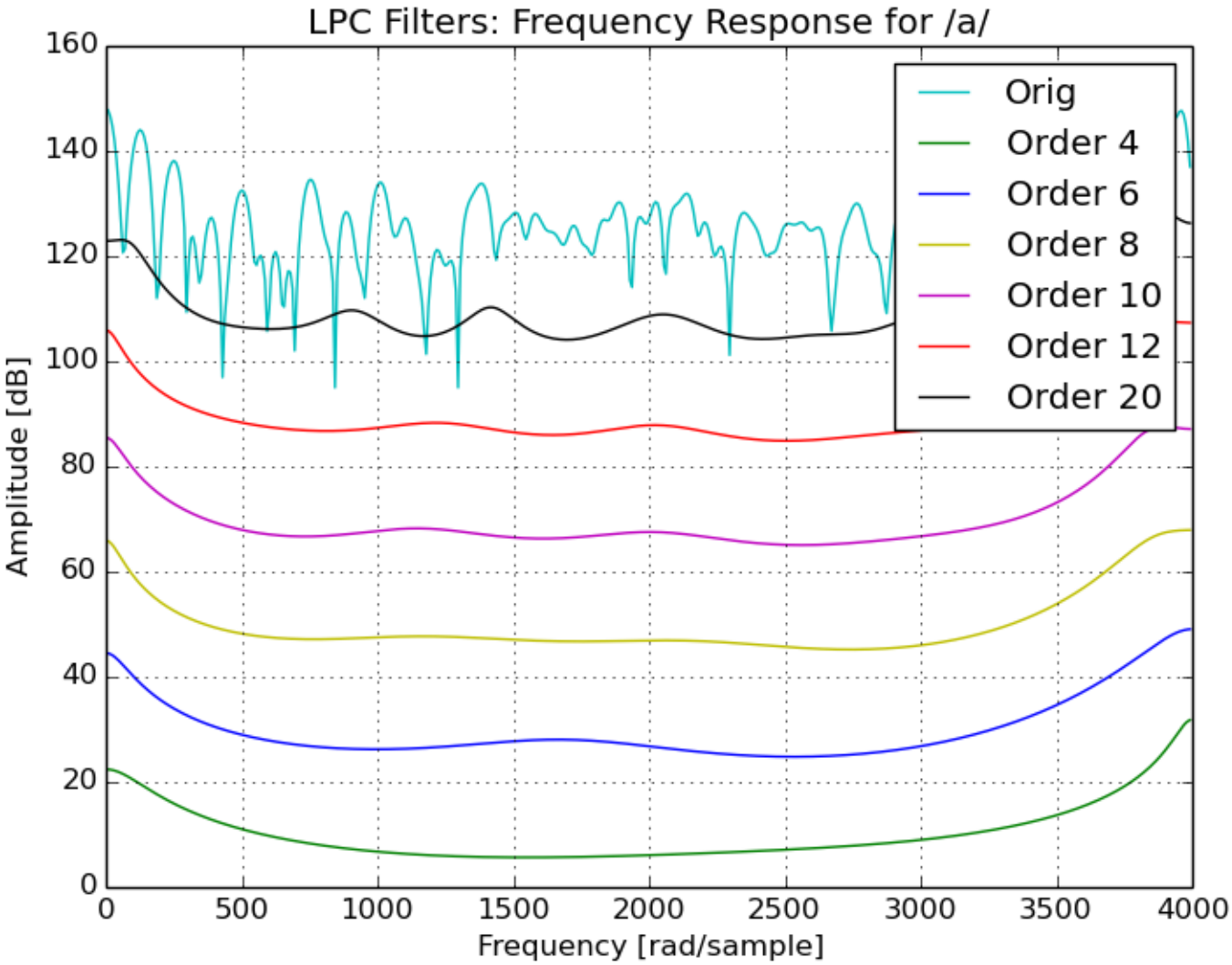
```
#from a ,b . a,b are calculated from LD algorithm
z, p, k = signal.tf2zpk(b, a)

#fig1 = 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.title('Pole / Zero Plot')
```

LPC spectrum magnitude

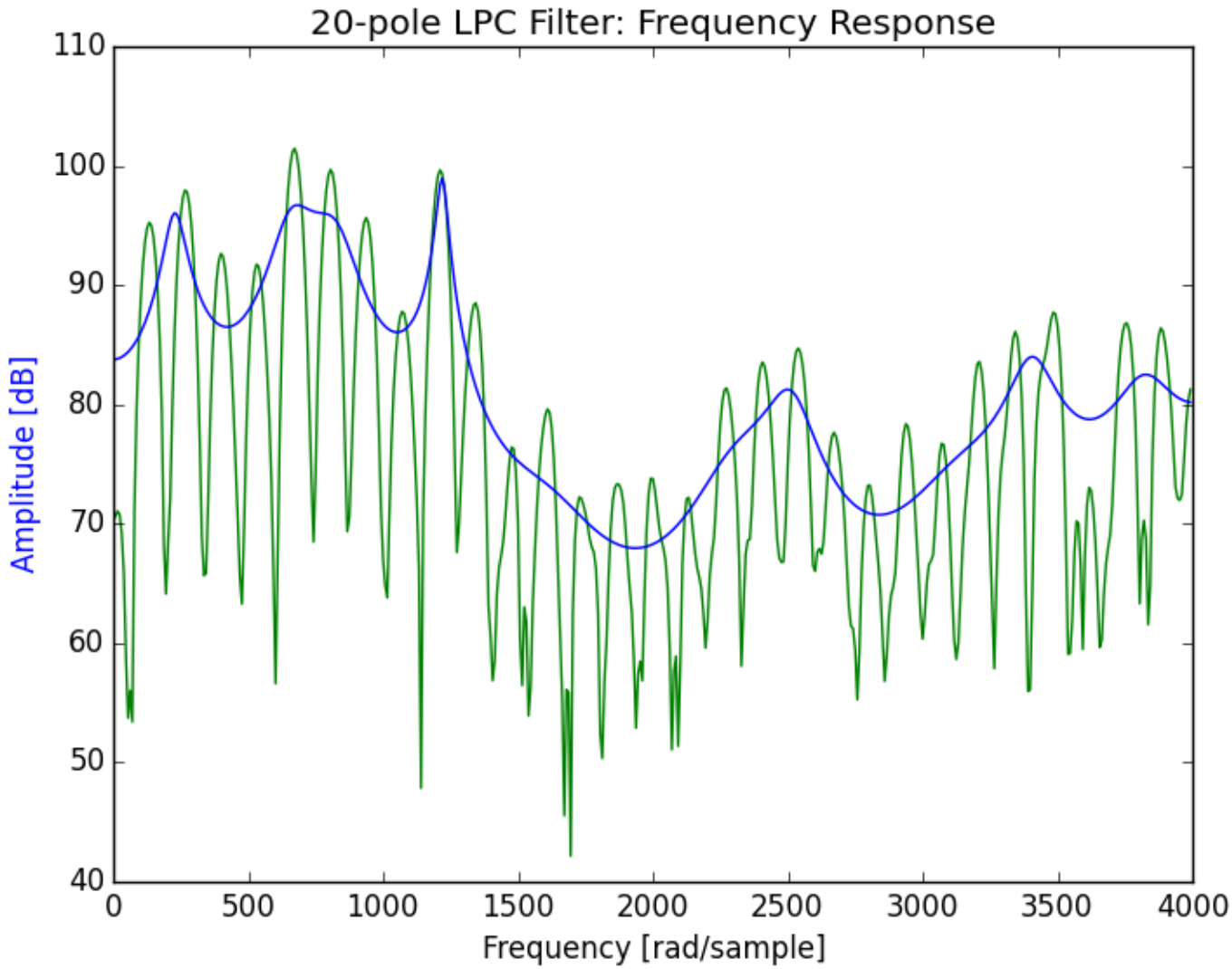
For /a/

All Order Filters

This has all the plots and every plot is shifted version of actual plot. The topmost plot corresponds to the original magnitude spectrum.

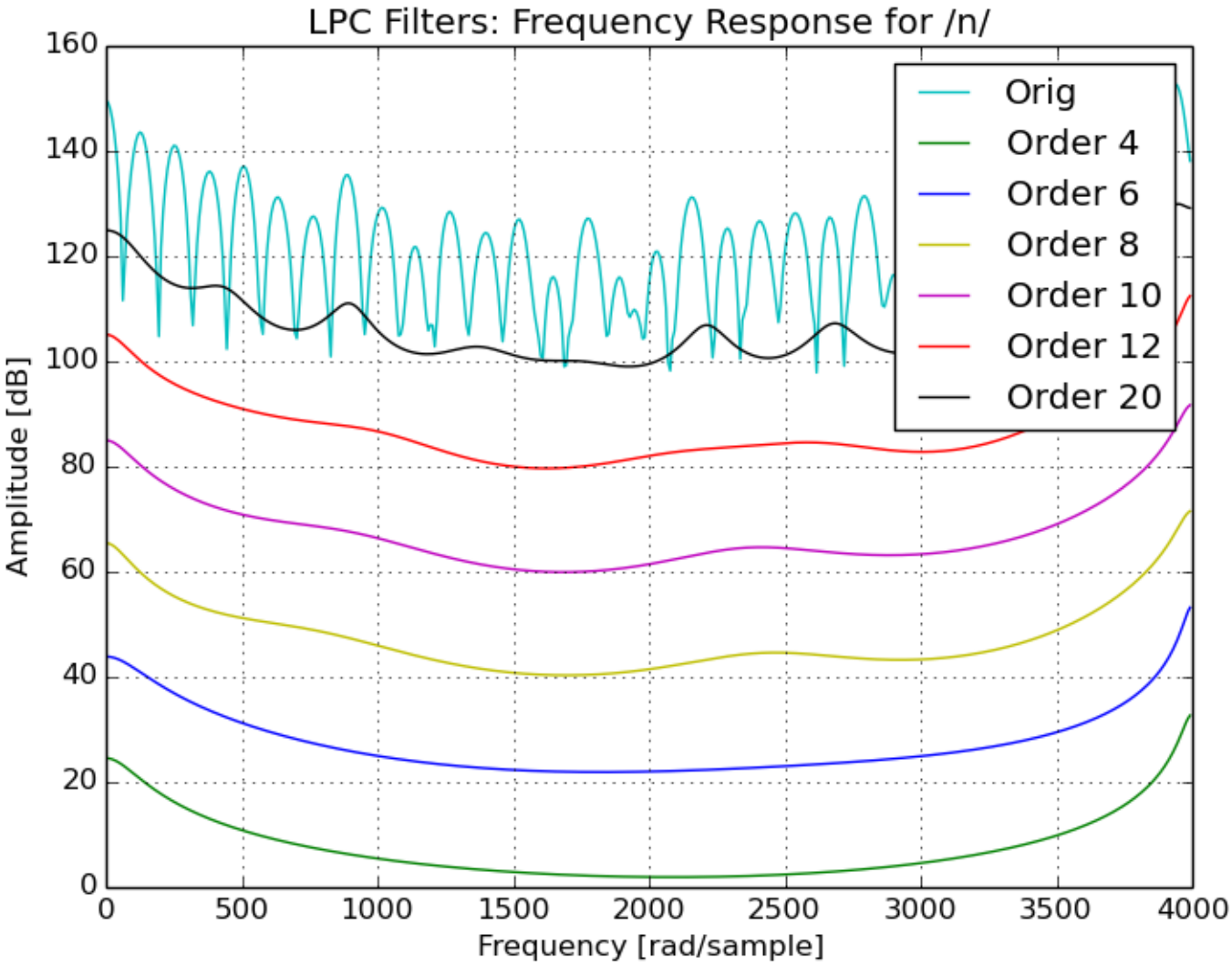


Superimposing order 20 plot on the narrowband dB magnitude spectrum. Original spectrum in Green and LPC spectrum in BLue.

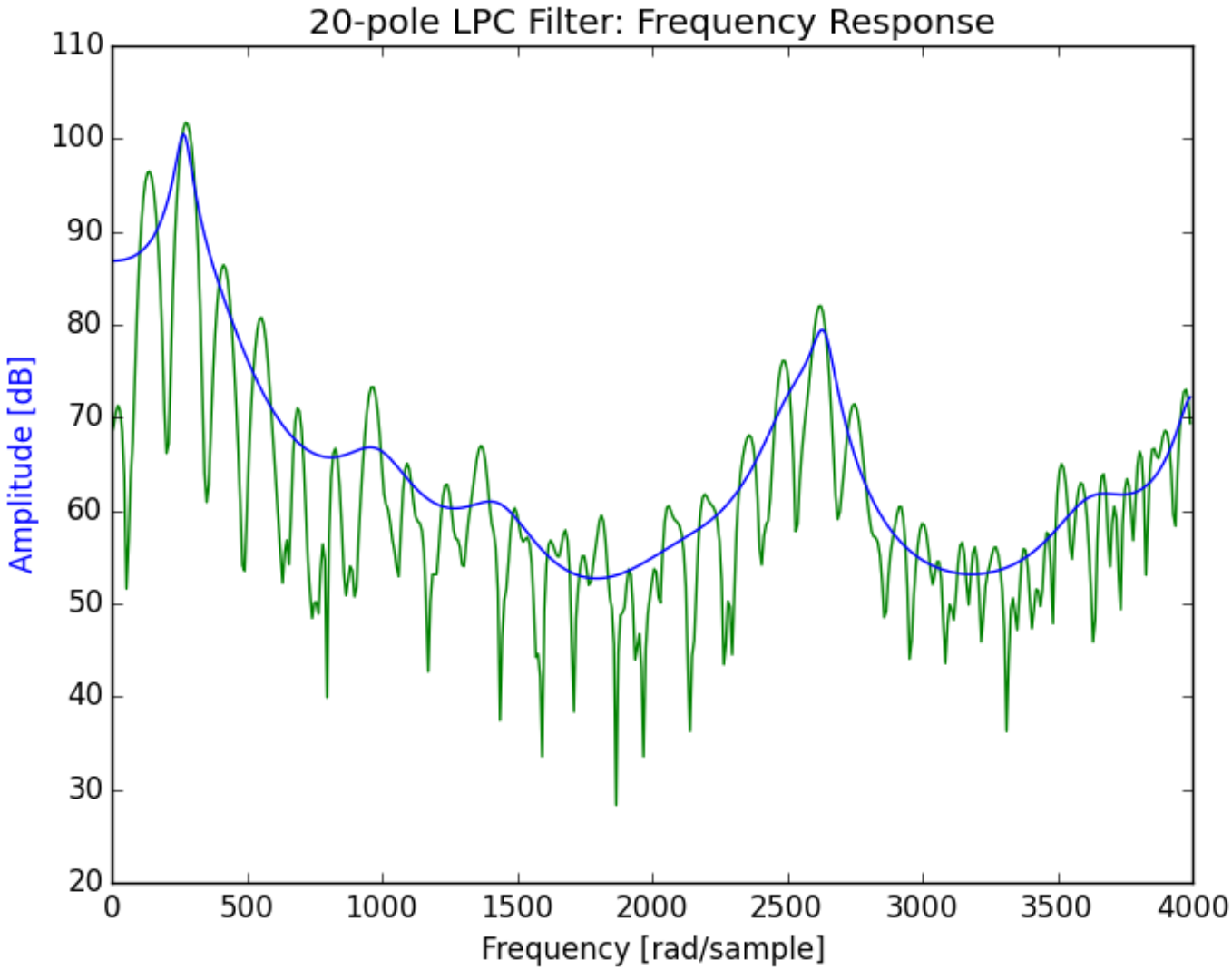


For /n/

All Order Filters

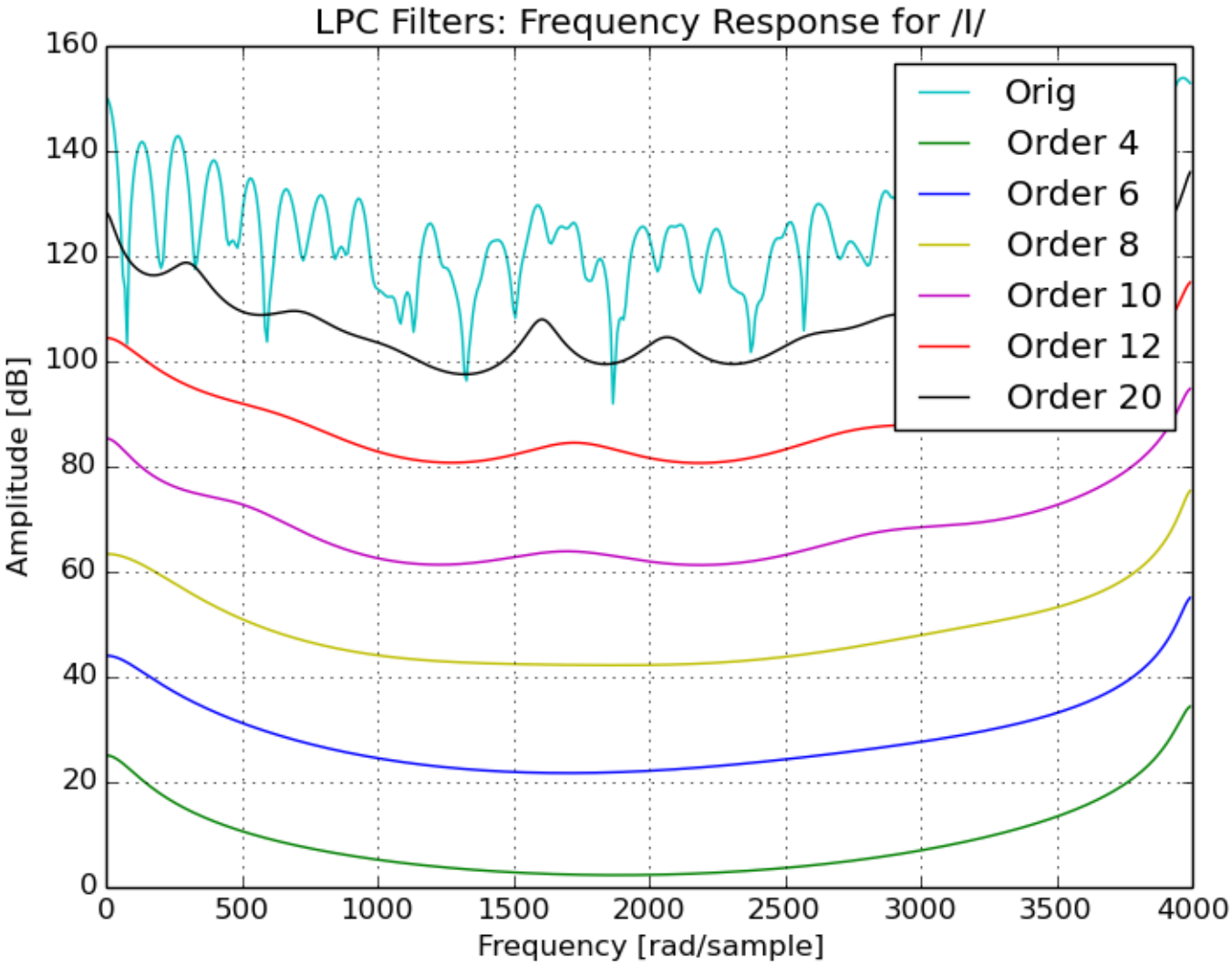


Superimposing order 20 plot on the narrowband dB magnitude spectrum.Original spectrum in Green and LPC spectrum in Blue. For /n/ we should increase the order because to model zeros we need a lot poles, hence approximation will be better.

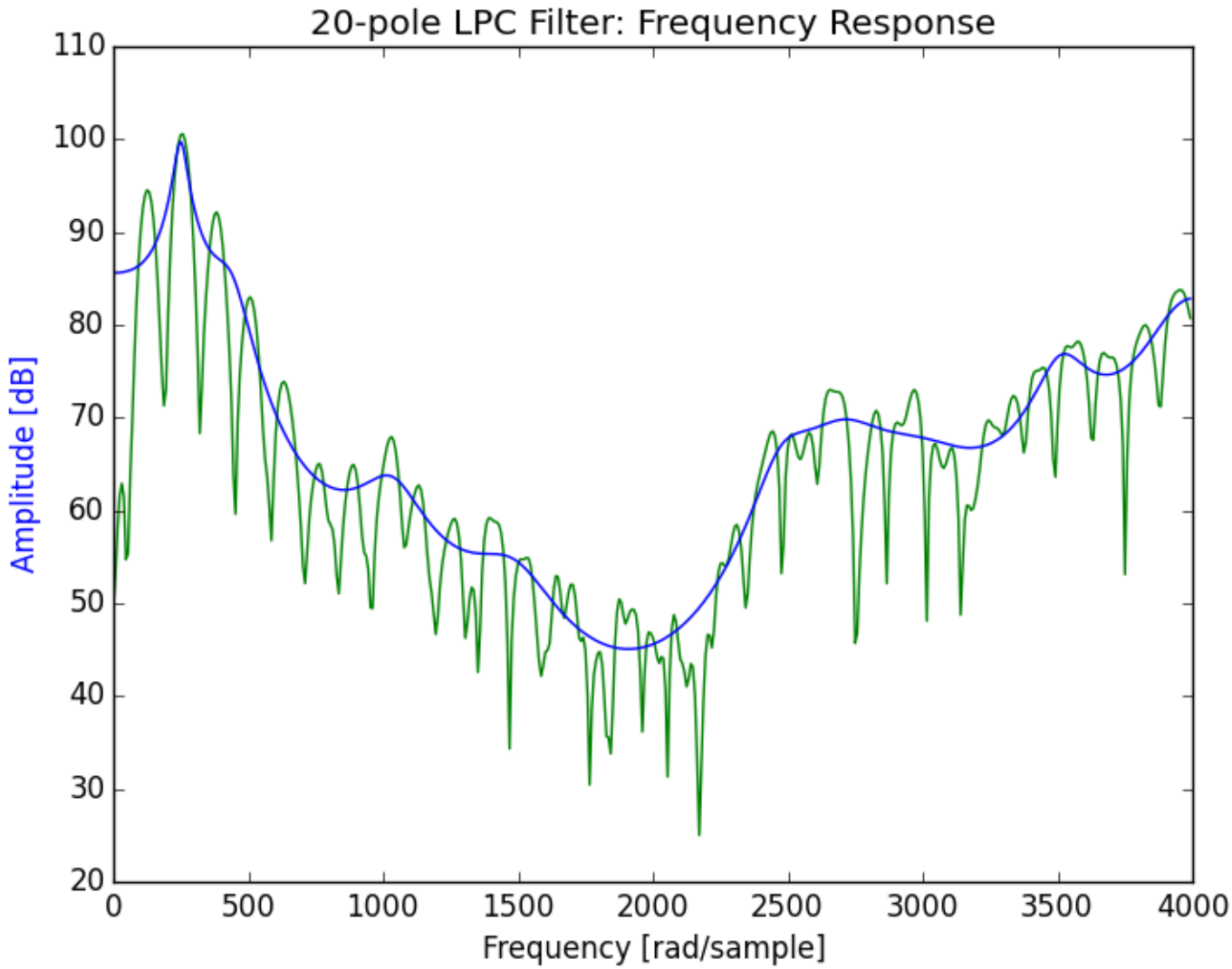


For /i/

All Order Filters

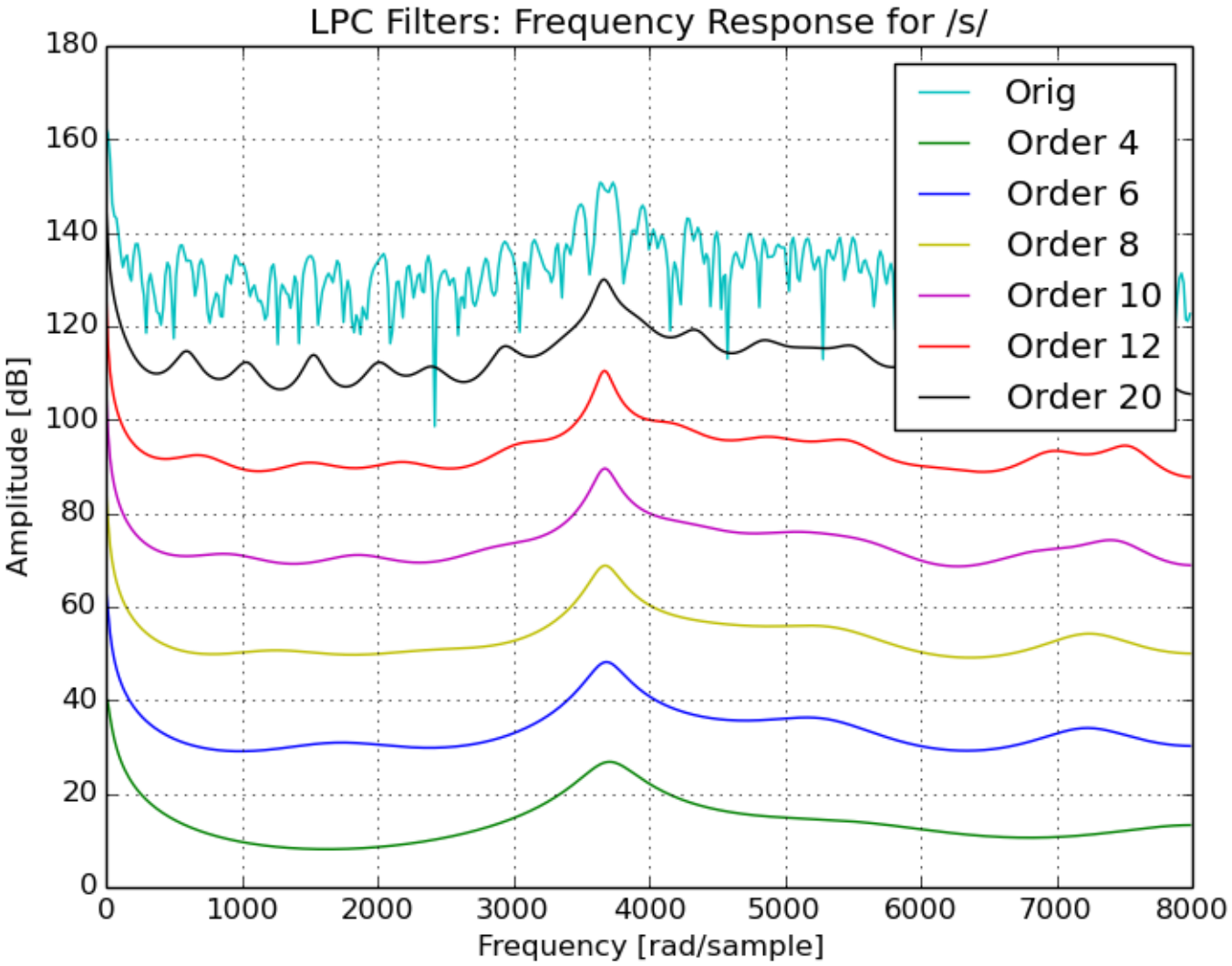


Superimposing order 20 plot on the narrowband dB magnitude spectrum.Original spectrum in Green and LPC spectrum in Blue.



For /s/

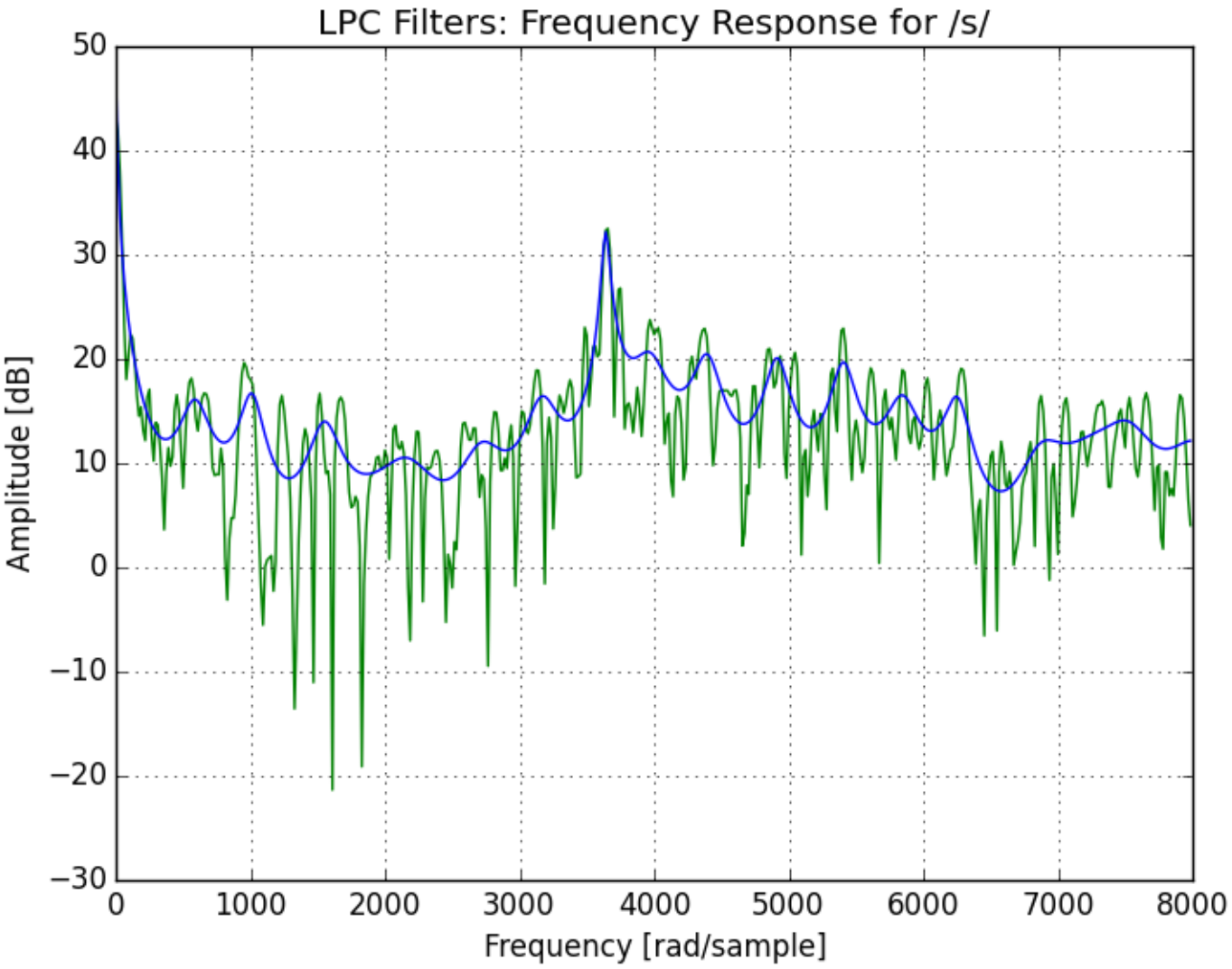
All Order Filters



Comments:

* Original signal has a lot of peaks in spectra. To gain such variability we should increase the LPC order.
* As we can see, going from bottom envelope to the top envelope, we are gaining variability in spectrum.
* LPC models the peaks in the spectrum better than its valleys.
* In the below plot, LPC order 40 and original magnitude spectrum has been plotted. Our LPC spectrum is close to the peak values than the valleys.
* For /s/ only, I have plotted with order 40 because sampling frequency is 16KHz. Hence, we will get better approximation.

Superimposing order 40 plot on the narrowband dB magnitude spectrum.Original spectrum in Green and LPC spectrum in Blue.




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

win_in = data[:Samp]*np.hamming(Samp)    #Samp = samplingfreq*windowTime
# Use Levinson's algorithm to calculate a[i]
for filterOrder in [2, 4, 6, 8, 10]:
    r = autocorr(win_inp) #autocorrelation of windowed input signal
    a = np.zeros(filterOrder+1)
    ao = np.zeros(a.shape)
    E = r[0]

    for i in range(1, filterOrder+1):
        k = 0
        Eo = E
        ao[1:len(a)] = a[1:len(a)]
        # finding K
        for l in range(1, i):
            k = k + ao[l]*r[i-l]
```

```
        k = (r[i] - k)/Eo

        # calculation of a's
        a[i] = k
        for l in range(1, i):
            a[l] = ao[l] - k*ao[i-l]
        E = (1-k*k)*Eo

a[0] = 1.0
a[1:len(a)] = -a[1:len(a)]
b = np.zeros(a.shape)
b[0] = np.sqrt(E)
G = b[0] # gain
#transfer function
w, h = signal.freqz(b, a)

# Frequency response of LPC filter
w, h = signal.freqz(b, a)
```

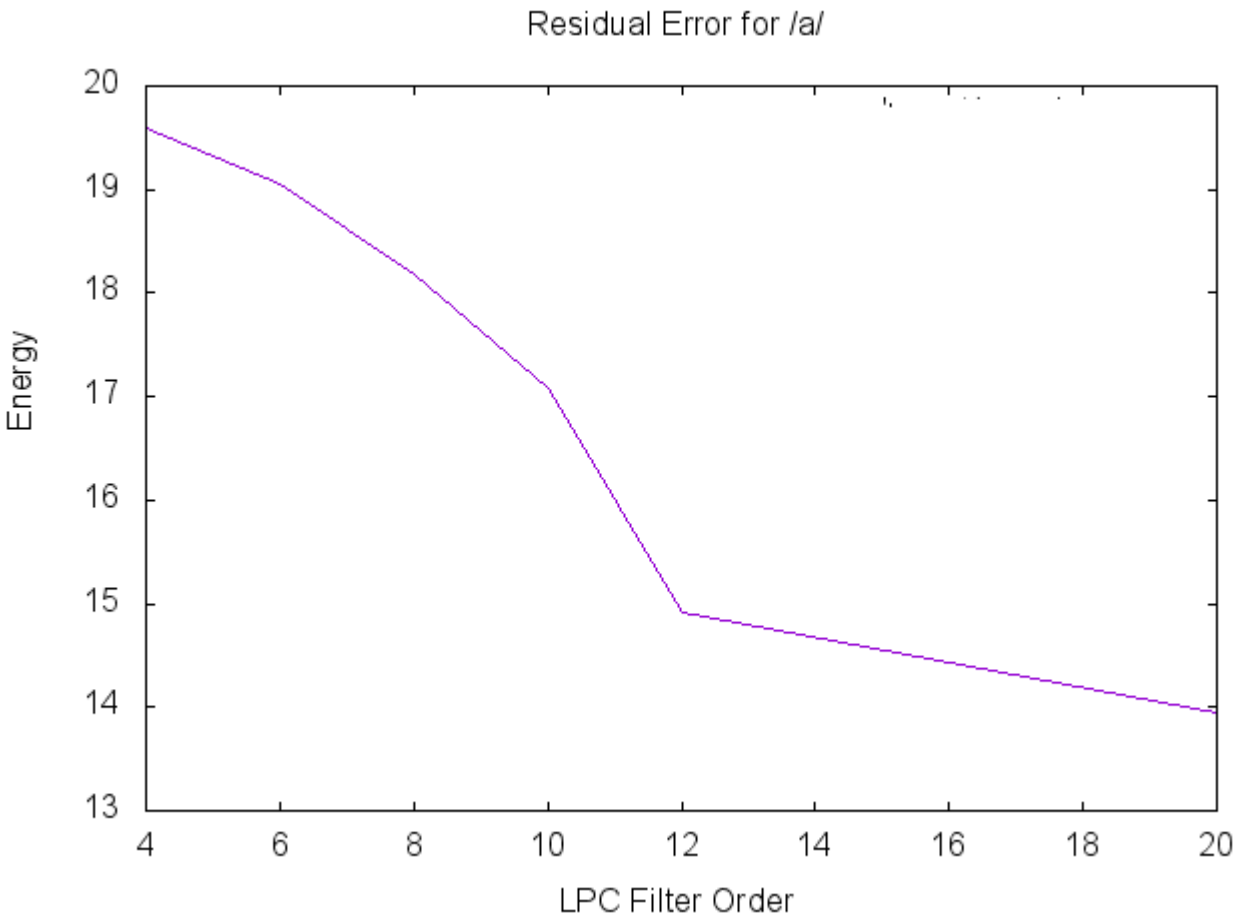
Error vs P Plots

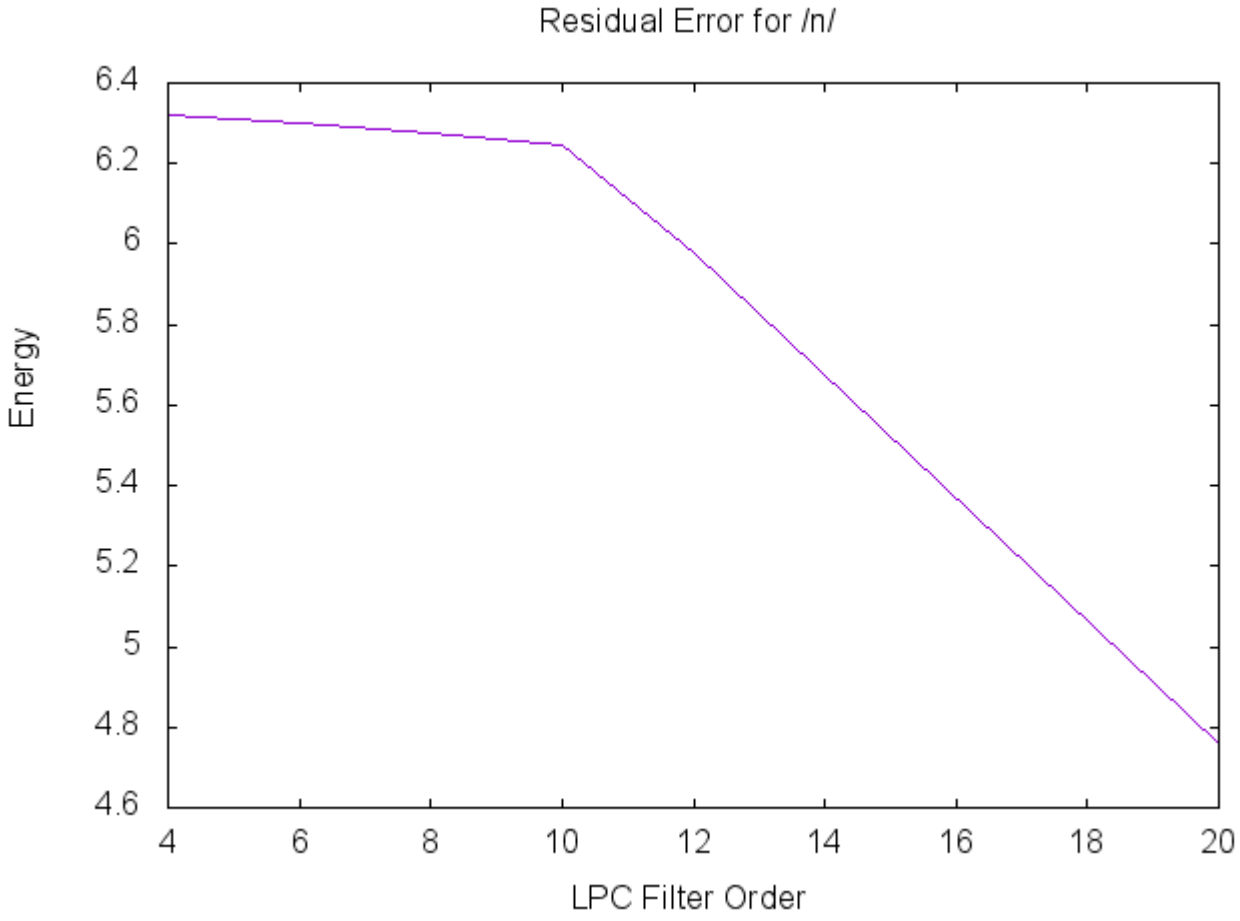
On the vertical axis, Energy is residual energy. As we can see, error signal energy is going down as we increase the LPC order.
Error is

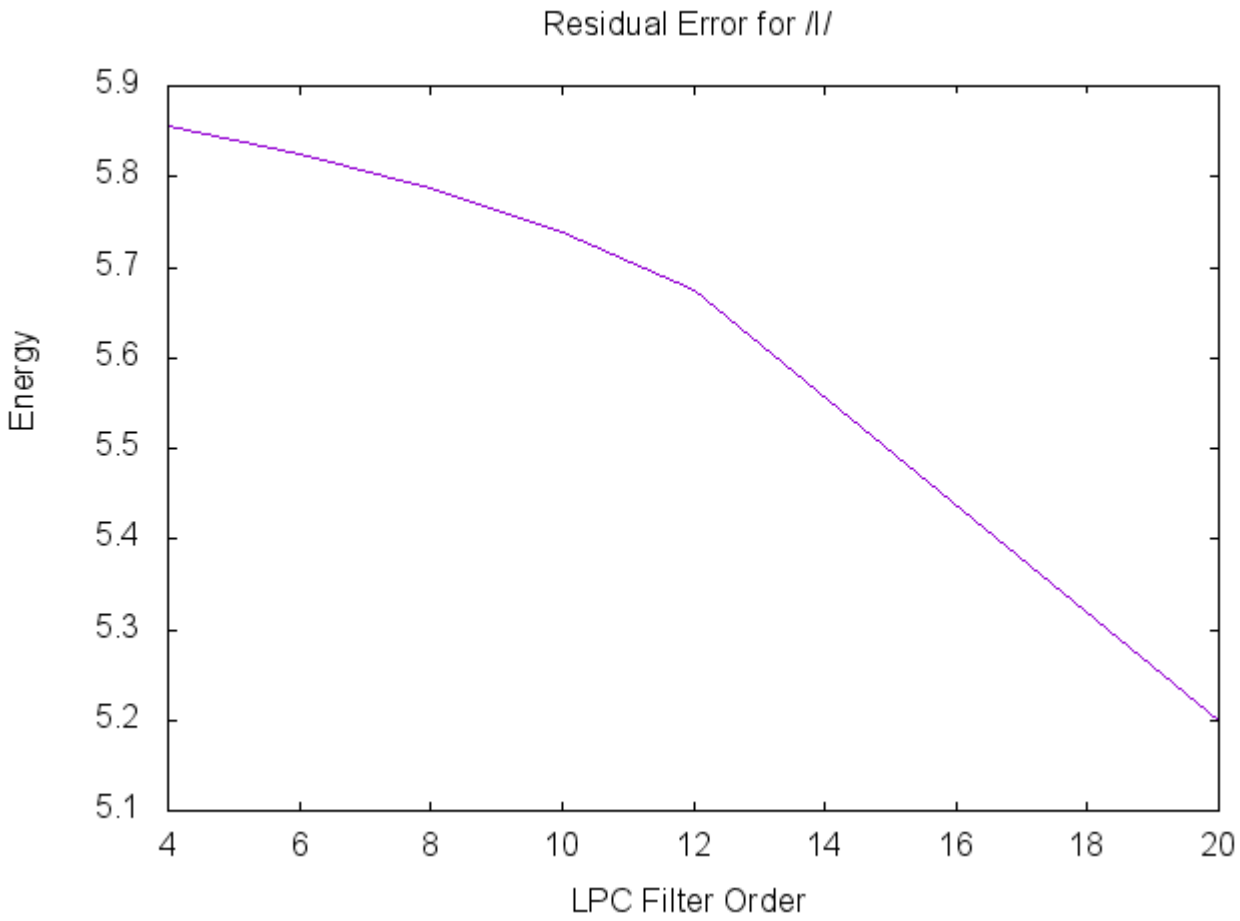
$$e(n) = s(n) + \sum_{k=1}^{p} a_k.s(n-k)$$

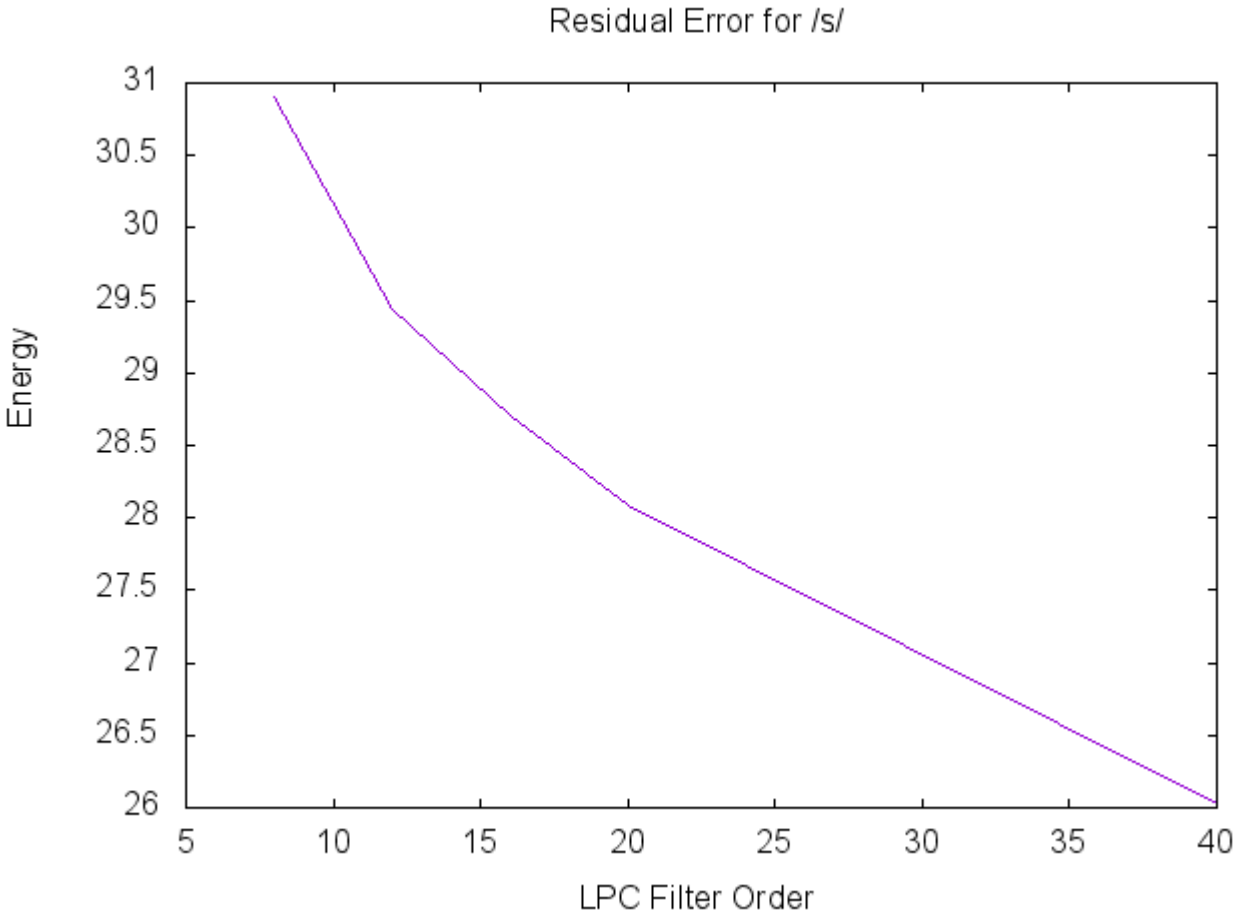
And E = energy is

$$E = \sum_{n=-\infty}^{\infty} e^2(n)$$




Residual Error for /n/
Energy
6.4
6.2
6
5.8
5.6
5.4
5.2
5
4.8
4.6
4
6
8
10
12
14
16
18
20
LPC Filter Order


Residual Error for /l/
Energy
5.9
5.8
5.7
5.6
5.5
5.4
5.3
5.2
5.1
4
6
8
10
12
14
16
18
20
LPC Filter Order

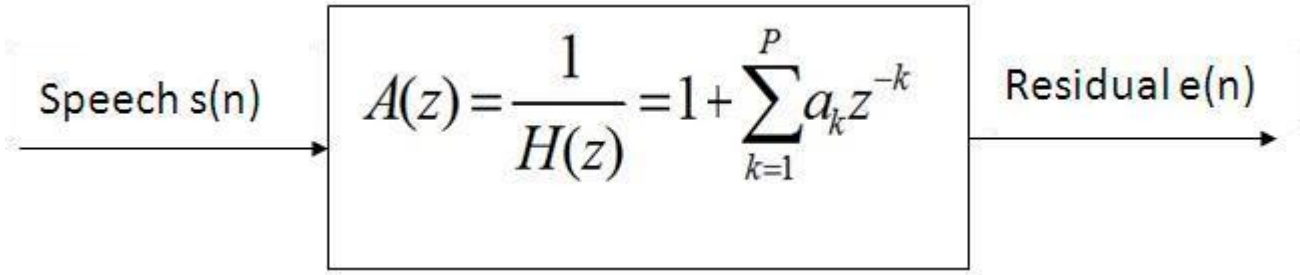


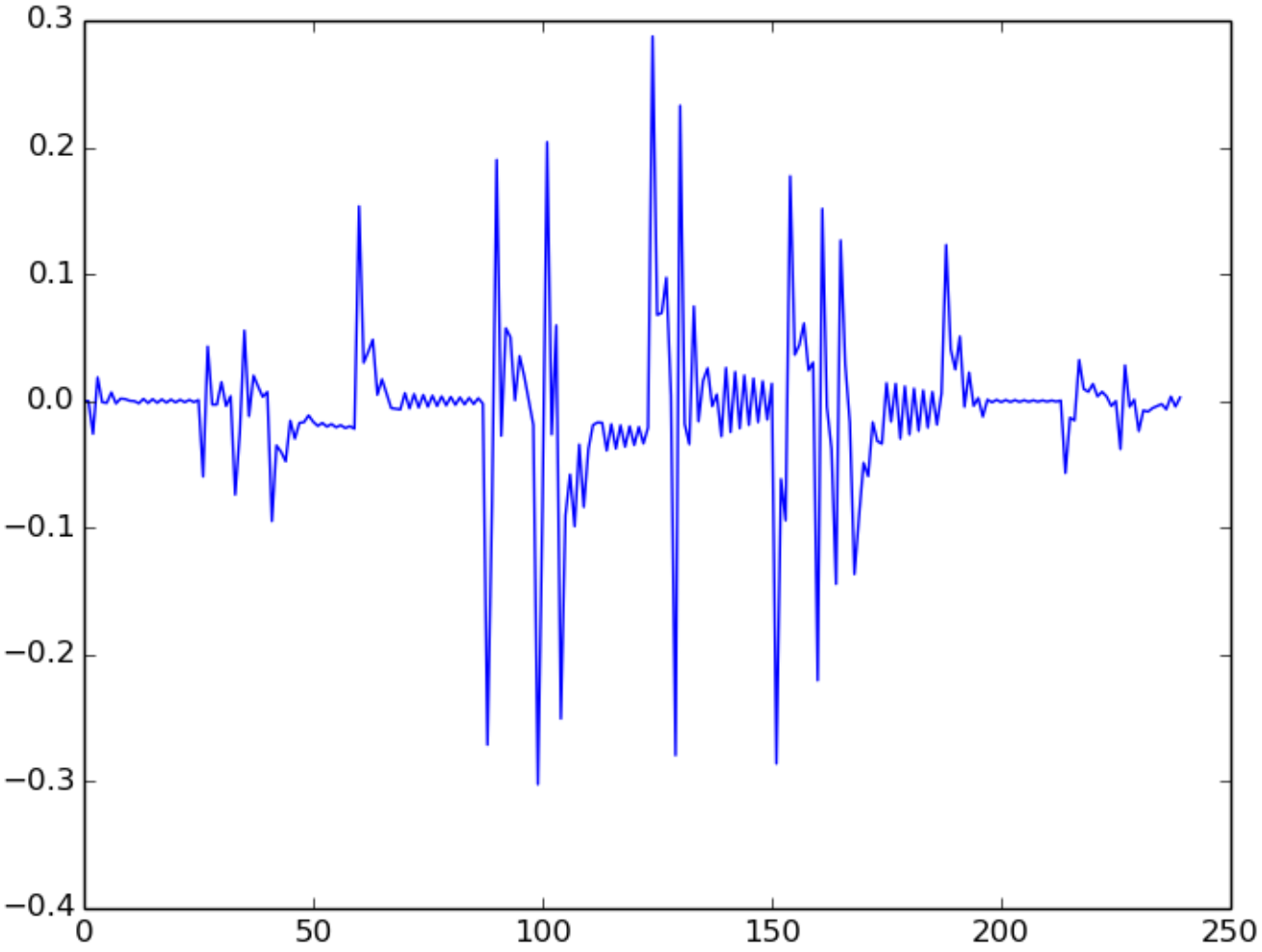
Comments:

* From the plots, /s/ has more error signal energy as compared to others. Hence unvoiced signal has more error signal energy than of voiced signal.

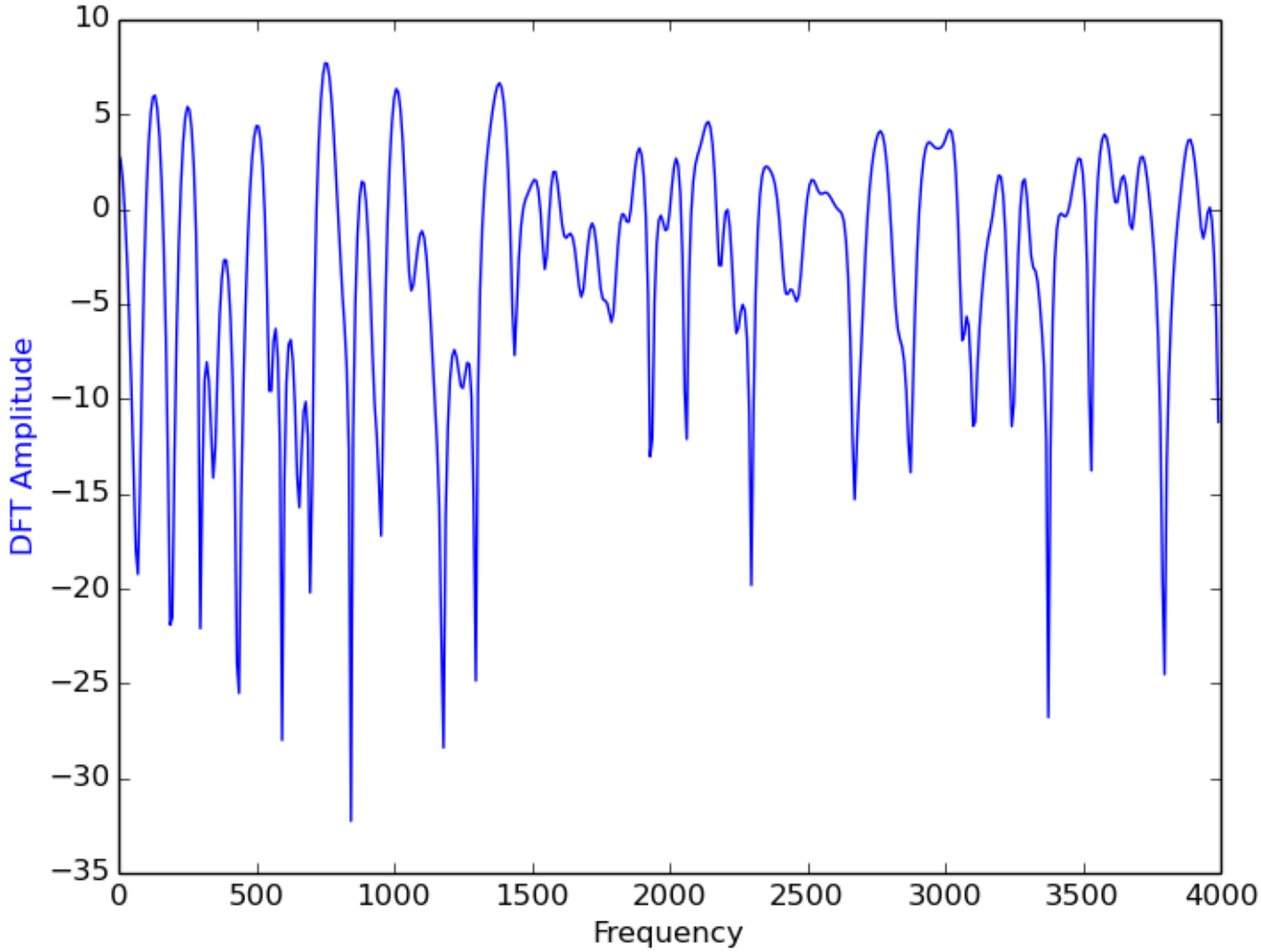
Inverse Filtering:

/a/
Residual error signal

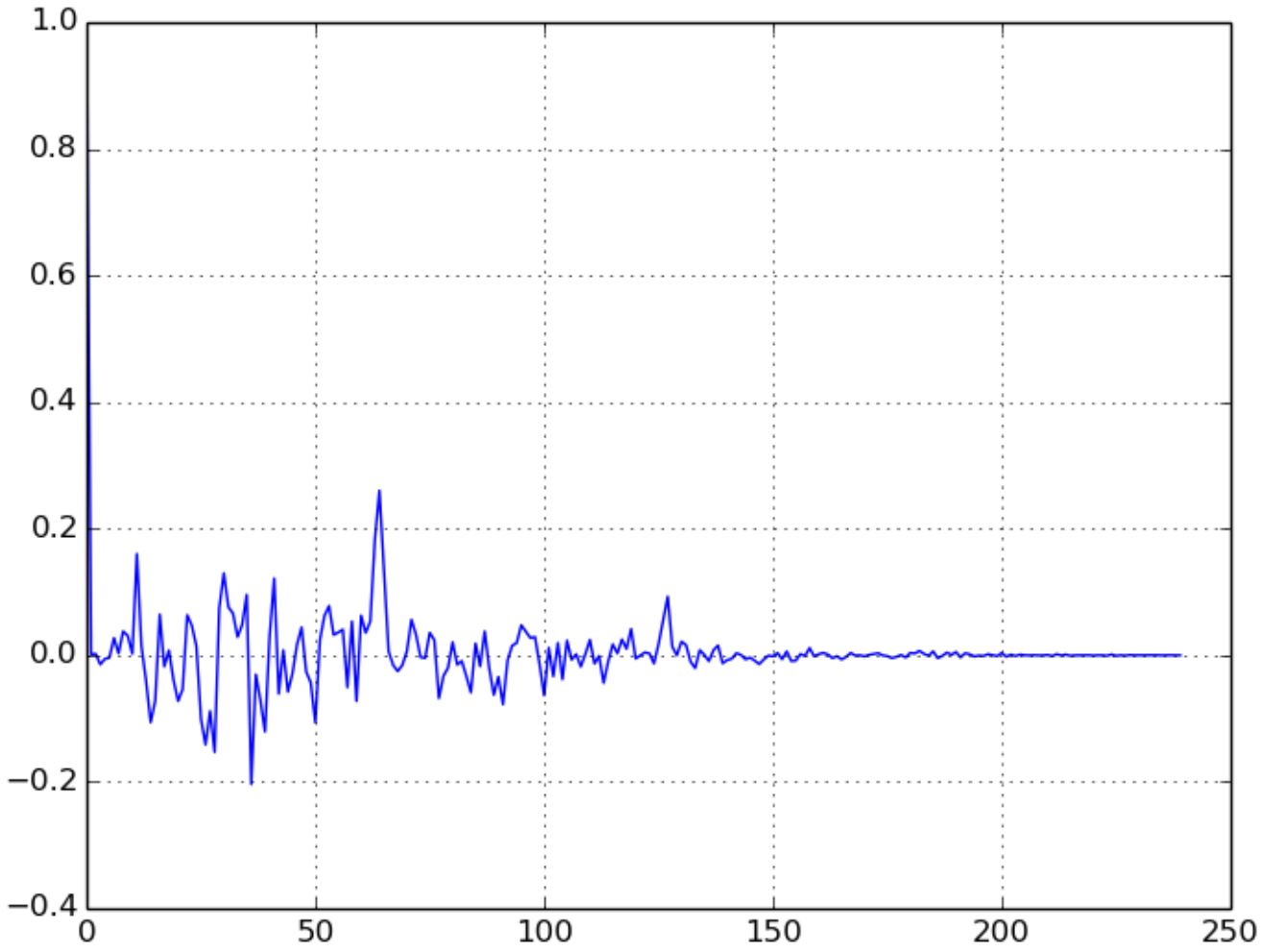




Magnitude Spectrum of the Residual signals



Residual autocorrelation plot



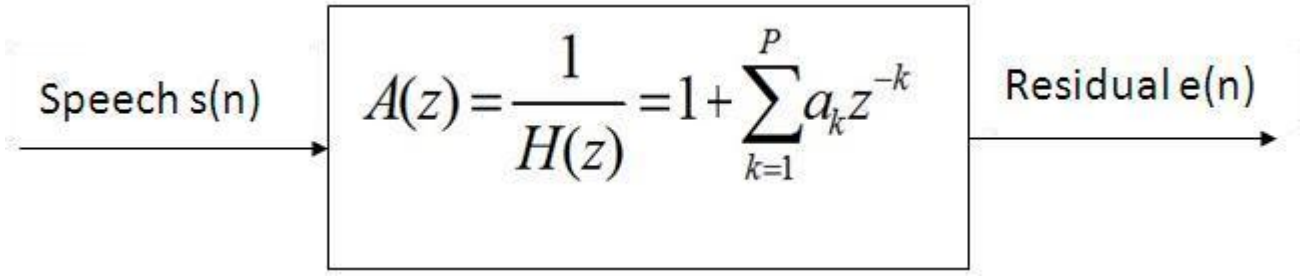
Comments:

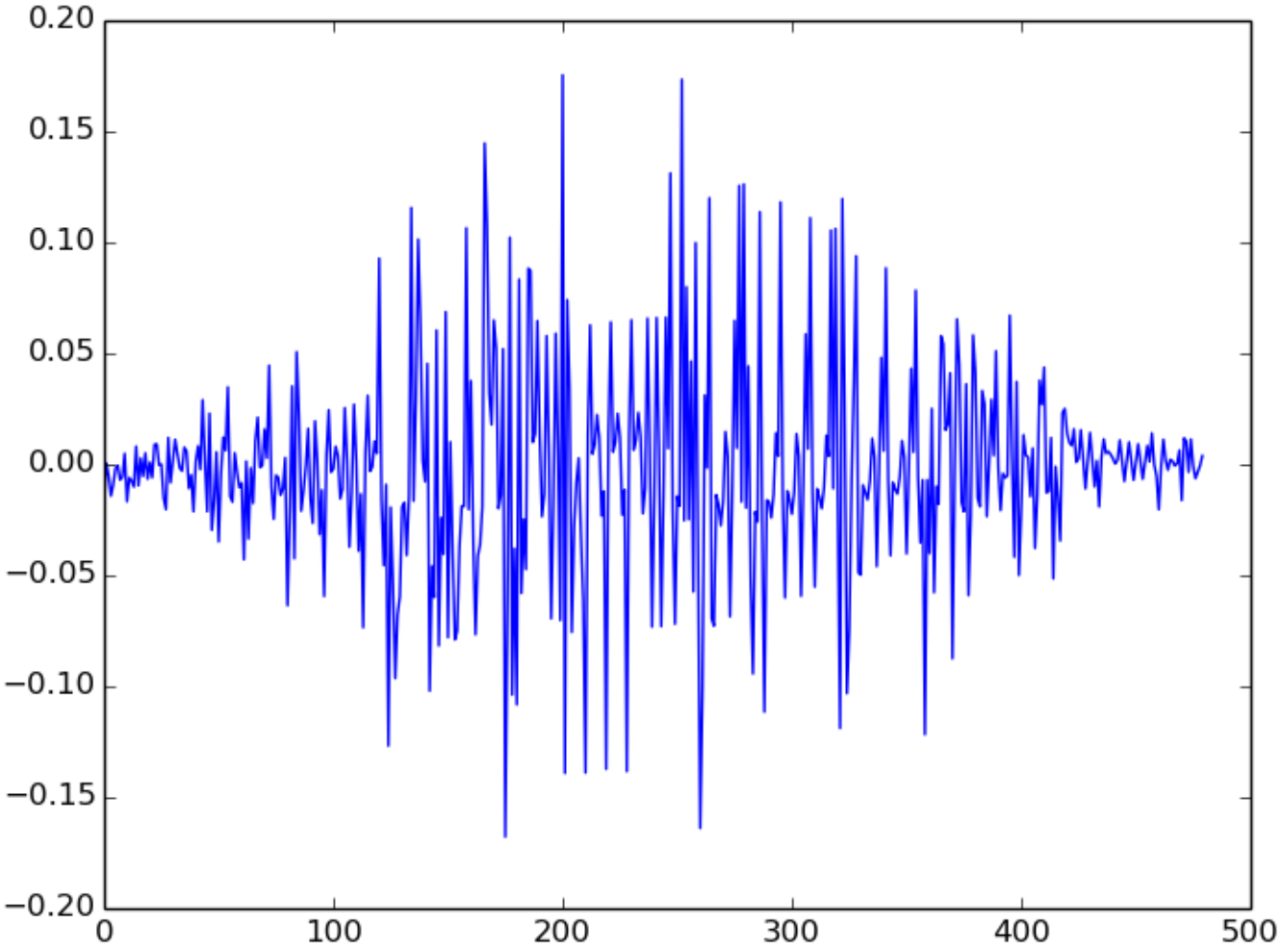
* As we observe the residual signal, it has peaks at few sample values. This is because at the (glottal) impluse we get more error.
* There is another reason for residual error, i.e tapering effect due to Hamming window. I haven't used Hamming window for this part.
* As we can see, there are lots of peaks in the plot, hence we can't always measure pitch period of the voiced sound from the residual waveform. But ideally, we should be able to measure in most of the cases.

** The autocorrelation function for /a/ has a peak at around 60 samples and the sampling frequency is 8kHz, hence the pitch of the signal is 8000/60 = 133.33 Hz. Hence, we can measure the pitch period from autocorrelation of the residual signal.
As seen from the autocorrelation function for the /s/ residual, we cannot calculate pitch from the /s/ sound because it is unvoiced

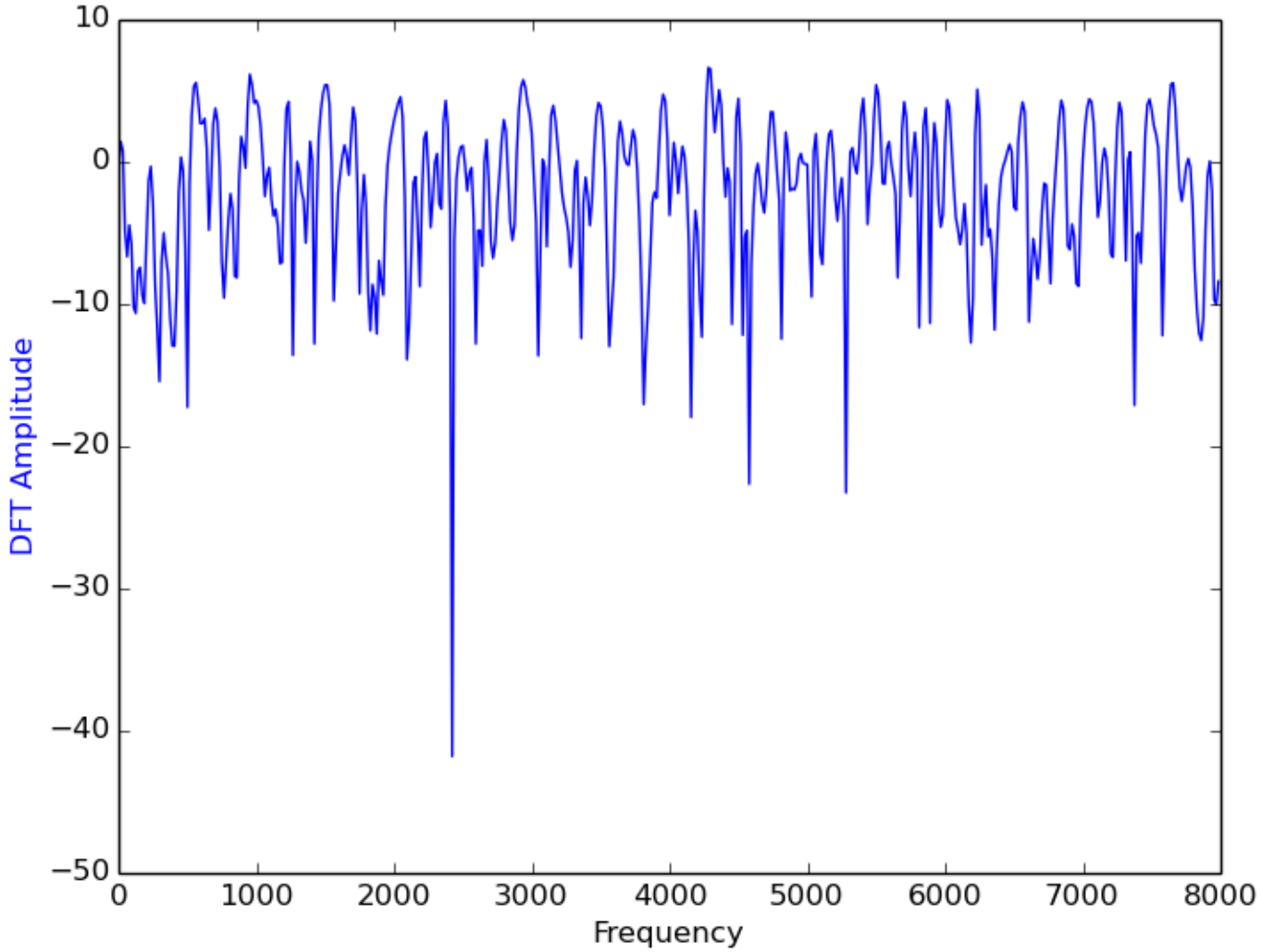
/s/

Residual error signal





Magnitude Spectrum of the Residual signals

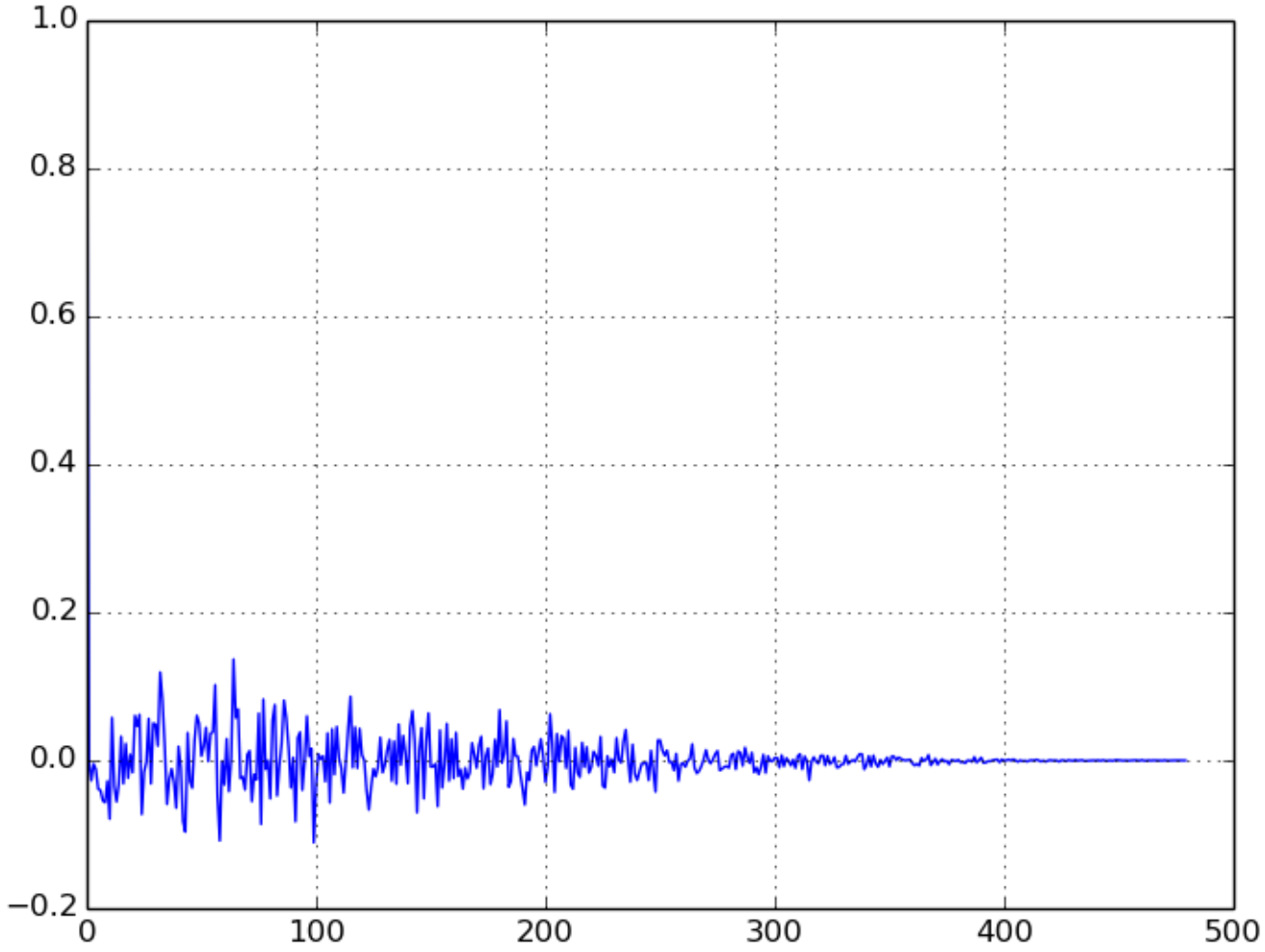


Comments:

*Residual error signal is changing very rapidly because /s/ is fricative. And the source is modelled as White Gaussian noise.

*As seen from the autocorrelation waveform for the /s/ residual, we cannot calculate pitch from the /s/ sound because it's unvoiced

Residual autocorrelation plot



Code Snippet for inverse filtering:

```
# use a,b from last part and w_in is the speech signal
res_signal = signal.lfilter(a, b, w_in)
#plt.plot(res_signal)
#plt.show()

#amplitude spectrum of residual signal
#dft1 = np.fft.fft(res_signal, 1024)
#freq1 = np.fft.fftfreq(dft1.shape[-1], 1/float(fSamp))
#plt.plot(freq1[:len(freq1)/2], 20*np.log10(np.abs(dft1[:len(dft1)/2])), 'b')

#for autocorrelation of residual signal
plt.plot(autocorr(res_signal))  # autocorr is defined in the 1st part
```