

# EE 679: Speech Processing Assignment 1

Kumar Ashutosh, 16D070043

August 31, 2019

## Question 1

### Reasoning

Given,

Formant Frequency = 900 Hz

Bandwidth = 200 Hz

Sampling Frequency = 16 kHz

We know from the Vocal Tract Model for Single Formant Resonator that roots of the transfer function is given by

$$r = e^{-B_i \pi T}, \theta = 2\pi F_i T$$

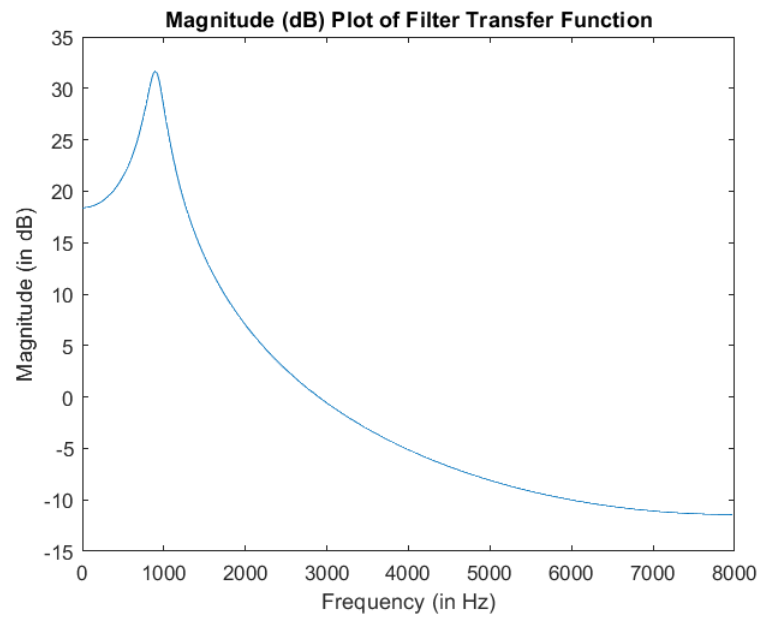
Using this and the given information we get the transfer function as

$$H(z) = \frac{1}{(1 - re^{j\theta} z^{-1})(1 - re^{-j\theta} z^{-1})}$$

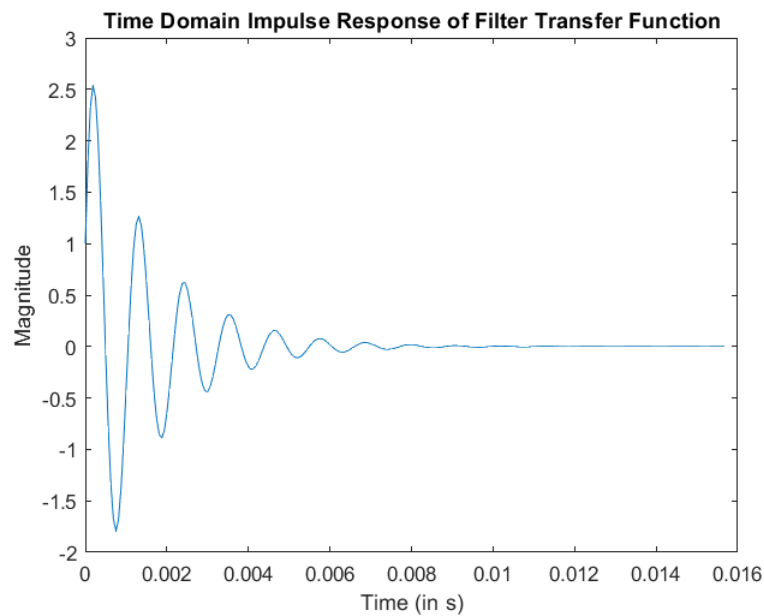
which on simplification becomes,

$$H(z) = \frac{1}{1 - 1.804z^{-1} + 0.9245z^{-2}}$$

The magnitude plot of this transfer function and the impulse response is plotted below.



**Magnitude Plot of the Transfer Function**



**Impulse Response of the Transfer Function**

## Code

### design\_tf.m

MATLAB function to design transfer function based on the roots of the filter.  
The roots are calculated as discussed in the above subsection.

```
1 function [Hz, num, den] = design_tf(formant_freq ,  
   bandwidth , samp_freq)  
2  
3 %% Finding roots of the Transfer Function  
4 r = exp(-bandwidth*pi*(1/samp_freq));  
5 theta = 2*pi*formant_freq*(1/samp_freq);  
6  
7 [~, num_formant] = size(formant_freq);  
8  
9 den = 1;  
10 for iter = 1:num_formant  
11     den = conv(den, [1, -2*r*cos(theta(iter)), r*r]);  
12 end  
13  
14 num = zeros(size(den));  
15 num(1) = 1; %Multiplying num and den by the highest  
   power of denominator  
16  
17 Hz = tf(num, den, 1/samp_freq);  
18 end
```

**question1.m** MATLAB main script to call the above function with the  
desired parameters and obtain the transfer function.

```
1 %% Data  
2 %Frequencies in Hertz  
3 formant_freq = 900;  
4 bandwidth = 200;  
5 samp_freq = 16e3;  
6  
7 %% Transfer Function  
8 [Hz, num, den] = design_tf(formant_freq, bandwidth,  
   samp_freq);  
9 [h, w] = freqz(num, den);
```

```

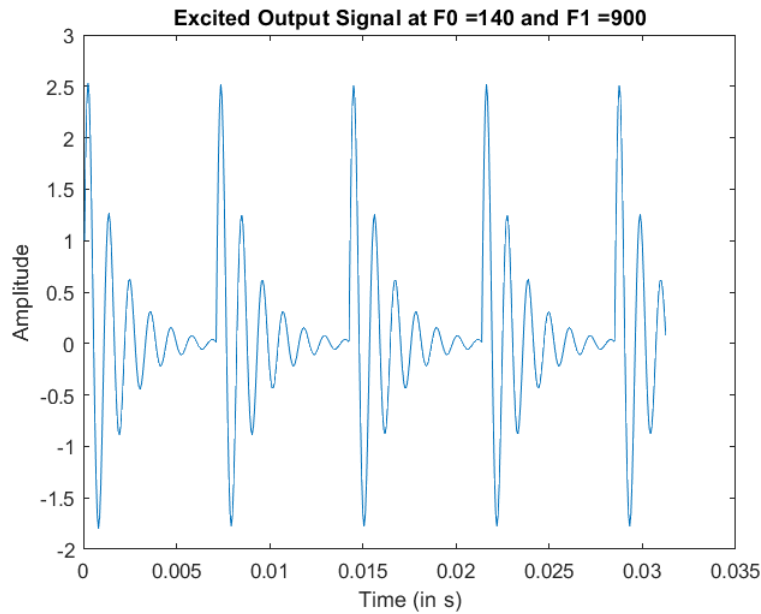
10
11 figure
12 plot((w*samp_freq)/(2*pi), 20*log10(abs(h)))
13 title('Magnitude (dB) Plot of Filter Transfer Function'
14 )
15 ylabel('Magnitude (in dB)')
16 xlabel('Frequency (in Hz)')
17
18 [val, index] = impz(num, den);
19
20 figure
21 plot(index/(samp_freq), val)
22 title('Time Domain Impulse Response of Filter Transfer
23 Function')
24 ylabel('Magnitude')
25 xlabel('Time (in s)')

```

## Question 2

### Reasoning

The periodic source excitation is modelled as a train of impulse at 140 Hz. The linear convolution is then done iteratively to find the output of the filter operation. The time domain output waveform is as attached below.



**Time Domain Filtering Output**

## 0.1 Code

### **filter\_input.m**

MATLAB function to perform time domain convolution

```

1 function output = filter_input(f0 , den , samp_freq ,
    duration)
2 %% Input Data
3 input_data = zeros(1, samp_freq*duration); %
    Corresponding to 0.5s data
4 input_data(1:round(samp_freq/f0):end) = 1;
5
6 %% Convolution Parameters
7 filter_coeffs = -den; %Coeffs are negative of each
    other except first one
8 filter_coeffs = filter_coeffs(2:end); %y[n-1] starts
    with second coeffs.
9 [~, filter_length] = size(filter_coeffs);
10
11 %% Time Domain Convolution

```

```

12 output = input_data;
13 [~, input_length] = size(input_data);
14
15 for iter=1:input_length
16     for filter_iter=1:filter_length
17         if iter - filter_iter < 1
18             add_factor = 0; %y[n] = 0 for n < 1
19         else
20             add_factor = output(iter-filter_iter);
21         end
22         output(iter) = output(iter) + add_factor*
            filter_coeffs(filter_iter);
23     end
24 end
25 end

```

#### question2.m

Main script to call the above function.

```

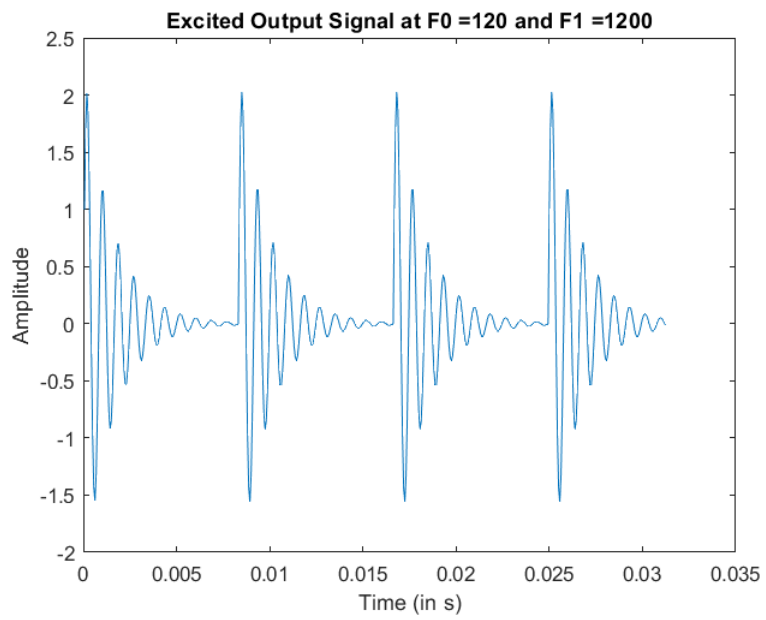
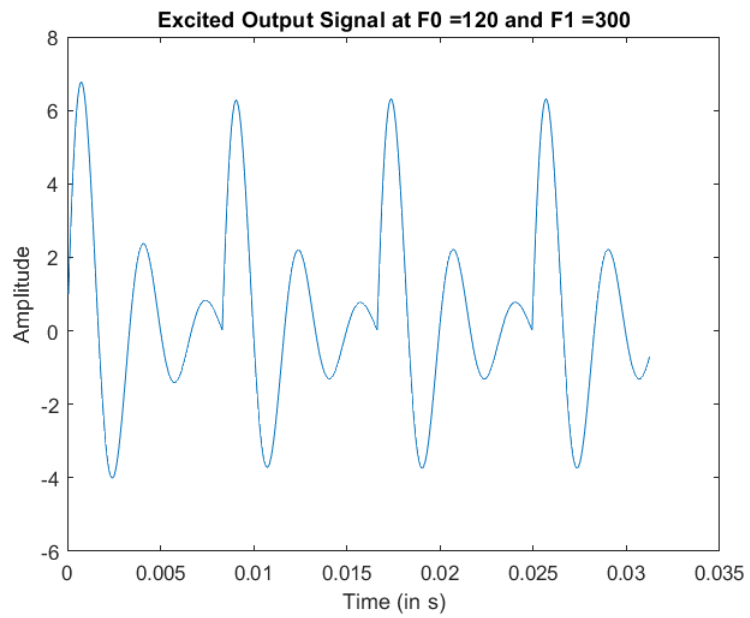
1 %% Parameters
2 %All frequencies in Hz
3 samp_freq = 16e3;
4 f0 = 140;
5 formant_freq = 900;
6 bandwidth = 200;
7 duration = 0.5; %in seconds
8
9 single_formant_analysis(formant_freq, f0, bandwidth,
    duration, samp_freq);

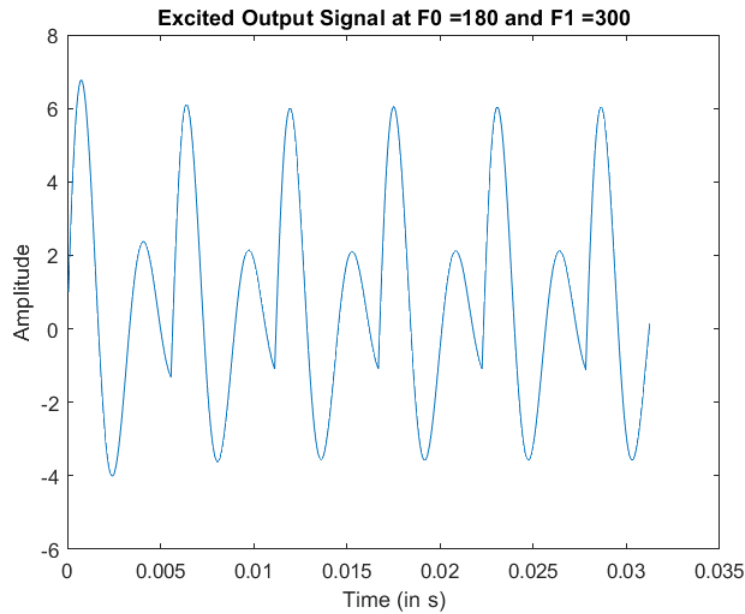
```

## Question 3

### Reasoning

For the three parts of this questions, the corresponding three waveform are as follows.





The three waveforms are also saved as .wav file and played to understand the sound. It is evident from the waveform as well that higher frequency of 1200 Hz results in faster oscillations in second waveform, hence the quality in this case decreases. (a) and (c) have the same formant frequency while F0 is same for (a) and (b). The waveforms in (a) and (c) looks similar due to the same formant frequency.

The three parts are also saved as audio files. Names of the audio files for each part are as follows :-

- (a) - output\_F0\_120\_F1\_300.wav
- (b) - output\_F0\_120\_F1\_1200.wav
- (c) - output\_F0\_180\_F1\_300.wav

## Code

### **single\_formant\_analysis.m**

MATLAB function to do Q1 and Q2 simultaneously with formant frequency.

```
1 function output = single_formant_analysis(formant_freq ,
    f0 , bandwidth , duration , samp_freq)
```



```

2
3
4 %% Transfer Function and Time Domain Output Signal
5 [~, ~, den] = design_tf(formant_freq, bandwidth,
    samp_freq);
6 output = filter_input(f0, den, samp_freq, duration);
7
8 %% Plotting over a few pitch periods
9 plot_samples = 500; %Plot 500 samples. 500 samples is
    considerably good
10 figure
11 plot((1:plot_samples)/samp_freq, output(1:plot_samples)
    );
12 title(strcat('Excited Output Signal at F0 = ', num2str(
    f0), ' and F1 = ', num2str(formant_freq)));
13 ylabel('Amplitude')
14 xlabel('Time (in s)')
15
16 %% Saving half second audio output
17 scaled_output = (2/(max(output(:))-min(output(:))))*(
    output - min(output(:))) - 1;
18 audiowrite(strcat('output_F0_', num2str(f0), '_F1_',
    num2str(formant_freq), '.wav'), scaled_output,
    samp_freq);
19
20 end

```

**question3.m** Main code to run the above function with the three desired parameters

```

1 %% Parameters
2
3 %All frequencies in Hz
4 samp_freq = 16e3;
5 duration = 0.5; %in seconds
6
7 %(a) F0 = 120 Hz, F1 = 300 Hz, B1 = 100 Hz
8 f0 = 120;
9 formant_freq = 300;

```

```

10 bandwidth = 100;
11 single_formant_analysis(formant_freq , f0 , bandwidth ,
    duration , samp_freq);
12
13 %(b) F0 = 120 Hz, F1=1200 Hz, B1 = 200 Hz
14 f0 = 120;
15 formant_freq = 1200;
16 bandwidth = 200;
17 single_formant_analysis(formant_freq , f0 , bandwidth ,
    duration , samp_freq);
18
19 %(c) F0 = 180 Hz, F1 = 300 Hz, B1 = 100 Hz
20 f0 = 180;
21 formant_freq = 300;
22 bandwidth = 100;
23 single_formant_analysis(formant_freq , f0 , bandwidth ,
    duration , samp_freq);

```

## Question 4

### Reasoning

For multiple formant frequencies the transfer functions gets multiplied, or in other words, the roots is the combination of all the formant frequencies. The three vowels are saved as audio files. The name of the audio files for each vowel are as follows :-

- "a" for 120Hz - a-120.wav
- "a" for 220Hz - a-220.wav
- "i" for 120Hz - i-120.wav
- "i" for 220Hz - i-220.wav
- "u" for 120Hz - u-120.wav
- "u" for 220Hz - u-220.wav

## Code

### **process\_and\_save\_audio.m**

MATLAB function to process the input as per the formant frequency, obtain the output, scale it accordingly and save it to a wav file.

```
1 function scaled_output = process_and_save_audio(f0 ,  
    formant_freq , bandwidth , samp_freq , save_audio ,  
    filename)  
2  
3 [~, ~, denominator] = design_tf(formant_freq , bandwidth  
    , samp_freq);  
4 output = filter_input(f0 , denominator , samp_freq , 0.5);  
5 %Scaling the output while saving to audio file  
6 scaled_output = (2/(max(output(:))-min(output(:))))*(  
    output - min(output(:))) - 1;  
7 if save_audio == 1  
8     audiowrite(filename , scaled_output , samp_freq);  
9 end  
10 end
```

**question4.m** Main code to run the above frequencies for the required cases.

```
1 %% Global Variables  
2 samp_freq = 16e3;  
3 bandwidth = 100;  
4  
5 %% /a/ at f0 = 120  
6 f0 = 120;  
7 formant_freq = [730 , 1090 , 2440];  
8 process_and_save_audio(f0 , formant_freq , bandwidth ,  
    samp_freq , 1 , 'a-120.wav');  
9  
10 %% /a/ at f0 = 220  
11 f0 = 220;  
12 formant_freq = [730 , 1090 , 2440];  
13 process_and_save_audio(f0 , formant_freq , bandwidth ,  
    samp_freq , 1 , 'a-220.wav');  
14
```

```

15 %% /i/ at f0 = 120
16 f0 = 120;
17 formant_freq = [270, 2290, 3010];
18 process_and_save_audio(f0, formant_freq, bandwidth,
    samp_freq, 1, 'i-120.wav');
19
20 %% /i/ at f0 = 220
21 f0 = 220;
22 formant_freq = [270, 2290, 3010];
23 process_and_save_audio(f0, formant_freq, bandwidth,
    samp_freq, 1, 'i-220.wav');
24
25 %% /u/ at f0 = 120
26 f0 = 120;
27 formant_freq = [300, 870, 2240];
28 process_and_save_audio(f0, formant_freq, bandwidth,
    samp_freq, 1, 'u-120.wav');
29
30 %% /u/ at f0 = 220
31 f0 = 220;
32 formant_freq = [300, 870, 2240];
33 process_and_save_audio(f0, formant_freq, bandwidth,
    samp_freq, 1, 'u-220.wav');

```

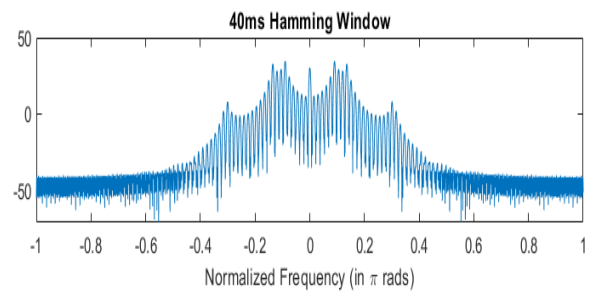
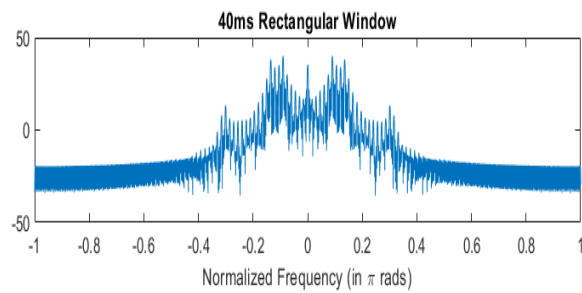
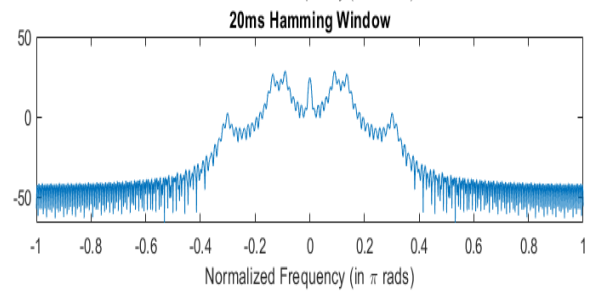
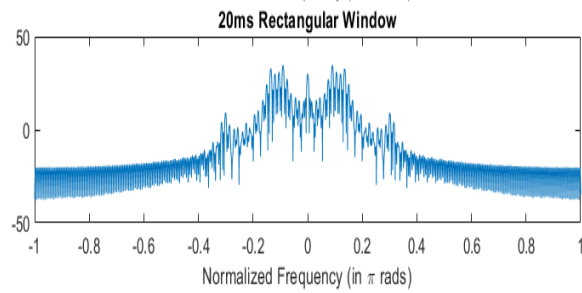
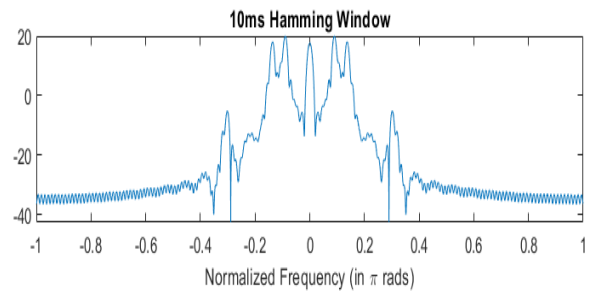
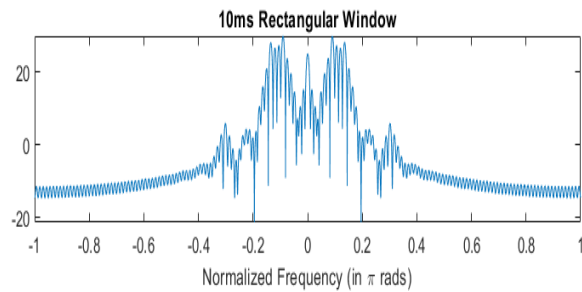
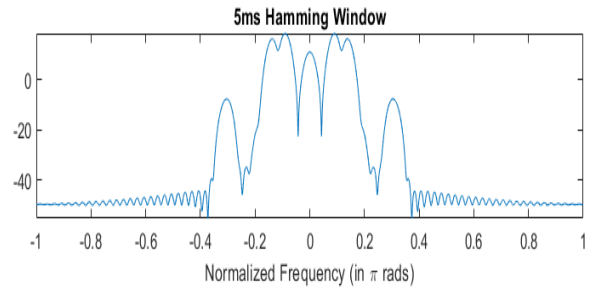
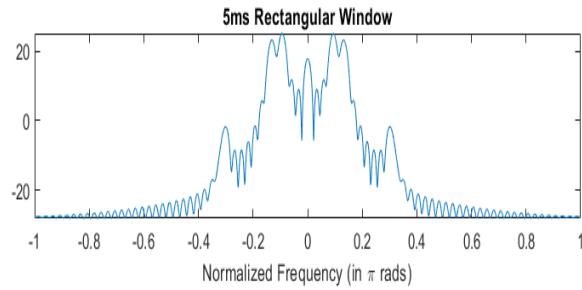
## Question 5

### Reasoning

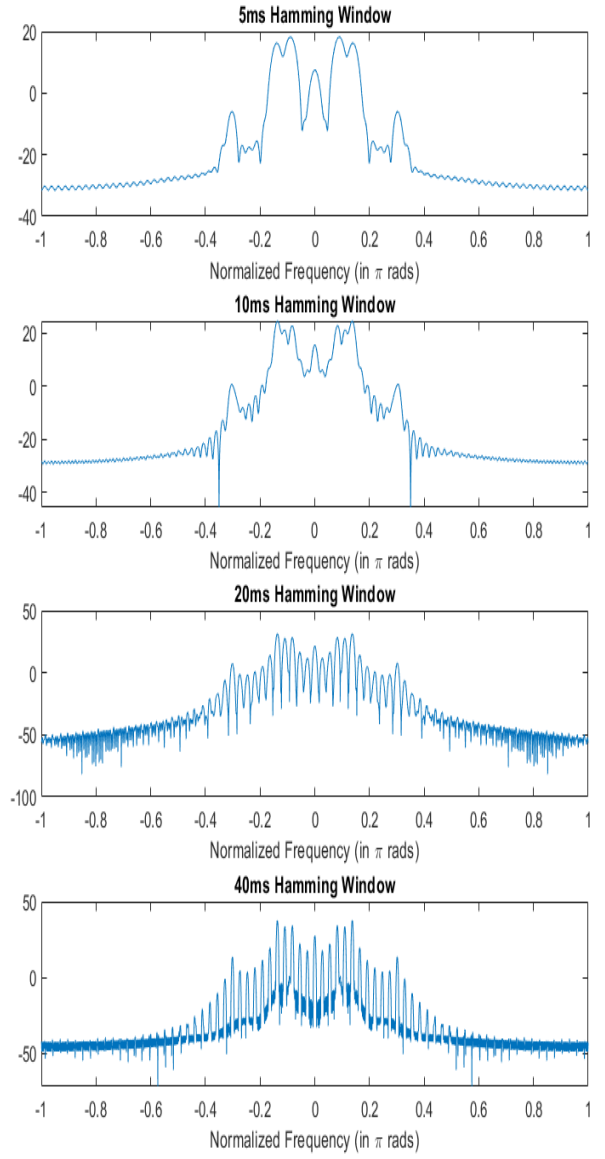
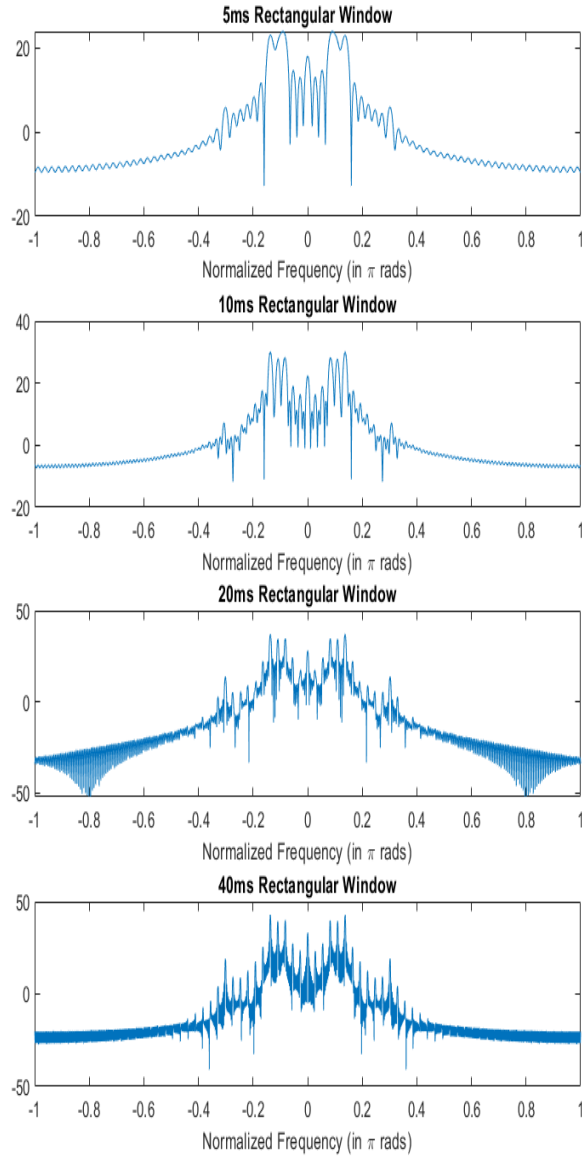
For both the window examples, waveform in the particular window was multiplied and other samples were let to be zero. FFT of the resulting signal is calculated and plotted in the normalized angular angle case.

The plots for the two F0 case is attached below -

/a/ at  $f_0 = 120$



/a/ at  $f_0 = 220$



(i) Comparison/Observations in the plots

- Different spacing in DFTF with different fundamental frequency F0.
- Increasing window length makes the output more clear and observable. For low window length, the frequencies are packed and not clear.
- Hamming Window vs Rectangular Frequency. In rectangular window, only first formant peak is observable, while in Hamming window other peaks are also clear. This can be attributed to the Energy Leakage due to the use of Rectangular Window.

## (ii) Estimation of Signal Parameters

- Estimated F0 - 117Hz and 223Hz (actual 120Hz and 220Hz)
- Estimated Formant Freq - 725Hz, 1095Hz and 2435Hz (actual 730Hz, 1090Hz, 2440Hz)

## Code

**compute\_dtft.m** MATLAB helper code to calculate the DTFT and return the log magnitude.

```

1 function dtft = compute_dtft(f0, formant_freq,
    bandwidth, samp_freq, window_size, window_type)
2 % Window type is 1 for rectangular and 2 for hamming
3
4 data = process_and_save_audio(f0, formant_freq,
    bandwidth, samp_freq, 0, '');
5 [~, data_points] = size(data);
6
7 if window_type == 1
8     window = ones(1, window_size);
9 else
10    window = hamming(window_size)';
11 end
12
13 windowed_data = zeros(size(data));
14 windowed_data(1:window_size) = window.*data(1:
    window_size);
15

```

```

16 dtfft = fft(windowed_data);
17 dtfft = circshift(20*log10(abs(dtfft)), data_points/2);
18
19 end

```

**question5.m** Main Code to run the above funtions on all the sixteen combinations and plot the output in a single plot.

```

1 %% Global Variables
2 samp_freq = 16e3;
3 bandwidth = 100;
4
5 %% /a/ at f0 = 120
6 f0 = 120;
7 formant_freq = [730, 1090, 2440];
8
9 figure
10
11 x_points = (1:8000)/4000 - 1;
12
13 % 5 ms Rectangular Window
14 window_type=1;
15 window_size=5e-3*samp_freq;
16 dtfft = compute_dtfft(f0, formant_freq, bandwidth,
    samp_freq, window_size, window_type);
17
18 subplot(4, 2, 1)
19 plot(x_points, dtfft)
20 title('5ms Rectangular Window')
21 xlabel('Normalized Frequency (in \pi rads)')
22
23 % 10 ms Rectangular Window
24 window_type=1;
25 window_size=10e-3*samp_freq;
26 dtfft = compute_dtfft(f0, formant_freq, bandwidth,
    samp_freq, window_size, window_type);
27
28 subplot(4, 2, 3)
29 plot(x_points, dtfft)

```



```

30 title('10ms Rectangular Window')
31 xlabel('Normalized Frequency (in \pi rads)')
32
33 % 20 ms Rectangular Window
34 window_type=1;
35 window_size=20e-3*samp_freq;
36 dtft = compute_dtft(f0, formant_freq, bandwidth,
    samp_freq, window_size, window_type);
37
38 subplot(4, 2, 5)
39 plot(x_points, dtft)
40 title('20ms Rectangular Window')
41 xlabel('Normalized Frequency (in \pi rads)')
42
43 % 40 ms Rectangular Window
44 window_type=1;
45 window_size=40e-3*samp_freq;
46 dtft = compute_dtft(f0, formant_freq, bandwidth,
    samp_freq, window_size, window_type);
47
48 subplot(4, 2, 7)
49 plot(x_points, dtft)
50 title('40ms Rectangular Window')
51 xlabel('Normalized Frequency (in \pi rads)')
52
53 % 5 ms Hamming Window
54 window_type=2;
55 window_size=5e-3*samp_freq;
56 dtft = compute_dtft(f0, formant_freq, bandwidth,
    samp_freq, window_size, window_type);
57
58 subplot(4, 2, 2)
59 plot(x_points, dtft)
60 title('5ms Hamming Window')
61 xlabel('Normalized Frequency (in \pi rads)')
62
63 % 10 ms Hamming Window
64 window_type=2;

```

```

65 window_size=10e-3*samp_freq;
66 dtft = compute_dtft(f0, formant_freq, bandwidth,
    samp_freq, window_size, window_type);
67
68 subplot(4, 2, 4)
69 plot(x_points, dtft)
70 title('10ms Hamming Window')
71 xlabel('Normalized Frequency (in \pi rads)')
72
73 % 20 ms Hamming Window
74 window_type=2;
75 window_size=20e-3*samp_freq;
76 dtft = compute_dtft(f0, formant_freq, bandwidth,
    samp_freq, window_size, window_type);
77
78 subplot(4, 2, 6)
79 plot(x_points, dtft)
80 title('20ms Hamming Window')
81 xlabel('Normalized Frequency (in \pi rads)')
82
83 % 40 ms Hamming Window
84 window_type=2;
85 window_size=40e-3*samp_freq;
86 dtft = compute_dtft(f0, formant_freq, bandwidth,
    samp_freq, window_size, window_type);
87
88 subplot(4, 2, 8)
89 plot(x_points, dtft)
90 title('40ms Hamming Window')
91 xlabel('Normalized Frequency (in \pi rads)')
92
93 sgtitle('/a/ at f0 = 120')
94
95
96
97 %% /a/ at f0 = 220
98 f0 = 220;
99 formant_freq = [730, 1090, 2440];

```

```

100
101 figure
102
103 x_points = (1:8000)/4000 - 1;
104
105 % 5 ms Rectangular Window
106 window_type=1;
107 window_size=5e-3*samp_freq;
108 dtft = compute_dtft(f0, formant_freq, bandwidth,
    samp_freq, window_size, window_type);
109
110 subplot(4, 2, 1)
111 plot(x_points, dtft)
112 title('5ms Rectangular Window')
113 xlabel('Normalized Frequency (in \pi rads)')
114
115 % 10 ms Rectangular Window
116 window_type=1;
117 window_size=10e-3*samp_freq;
118 dtft = compute_dtft(f0, formant_freq, bandwidth,
    samp_freq, window_size, window_type);
119
120 subplot(4, 2, 3)
121 plot(x_points, dtft)
122 title('10ms Rectangular Window')
123 xlabel('Normalized Frequency (in \pi rads)')
124
125 % 20 ms Rectangular Window
126 window_type=1;
127 window_size=20e-3*samp_freq;
128 dtft = compute_dtft(f0, formant_freq, bandwidth,
    samp_freq, window_size, window_type);
129
130 subplot(4, 2, 5)
131 plot(x_points, dtft)
132 title('20ms Rectangular Window')
133 xlabel('Normalized Frequency (in \pi rads)')
134

```

```

135 % 40 ms Rectangular Window
136 window_type=1;
137 window_size=40e-3*samp_freq;
138 dtft = compute_dtft(f0, formant_freq, bandwidth,
    samp_freq, window_size, window_type);
139
140 subplot(4, 2, 7)
141 plot(x_points, dtft)
142 title('40ms Rectangular Window')
143 xlabel('Normalized Frequency (in \pi rads)')
144
145 % 5 ms Hamming Window
146 window_type=2;
147 window_size=5e-3*samp_freq;
148 dtft = compute_dtft(f0, formant_freq, bandwidth,
    samp_freq, window_size, window_type);
149
150 subplot(4, 2, 2)
151 plot(x_points, dtft)
152 title('5ms Hamming Window')
153 xlabel('Normalized Frequency (in \pi rads)')
154
155 % 10 ms Hamming Window
156 window_type=2;
157 window_size=10e-3*samp_freq;
158 dtft = compute_dtft(f0, formant_freq, bandwidth,
    samp_freq, window_size, window_type);
159
160 subplot(4, 2, 4)
161 plot(x_points, dtft)
162 title('10ms Hamming Window')
163 xlabel('Normalized Frequency (in \pi rads)')
164
165 % 20 ms Hamming Window
166 window_type=2;
167 window_size=20e-3*samp_freq;
168 dtft = compute_dtft(f0, formant_freq, bandwidth,
    samp_freq, window_size, window_type);

```

```

169
170 subplot(4, 2, 6)
171 plot(x_points, dtft)
172 title('20ms Hamming Window')
173 xlabel('Normalized Frequency (in \pi rads)')
174
175 % 40 ms Hamming Window
176 window_type=2;
177 window_size=40e-3*samp_freq;
178 dtft = compute_dtft(f0, formant_freq, bandwidth,
    samp_freq, window_size, window_type);
179
180 subplot(4, 2, 8)
181 plot(x_points, dtft)
182 title('40ms Hamming Window')
183 xlabel('Normalized Frequency (in \pi rads)')
184
185 sgtitle('/a/ at f0 = 220')

```