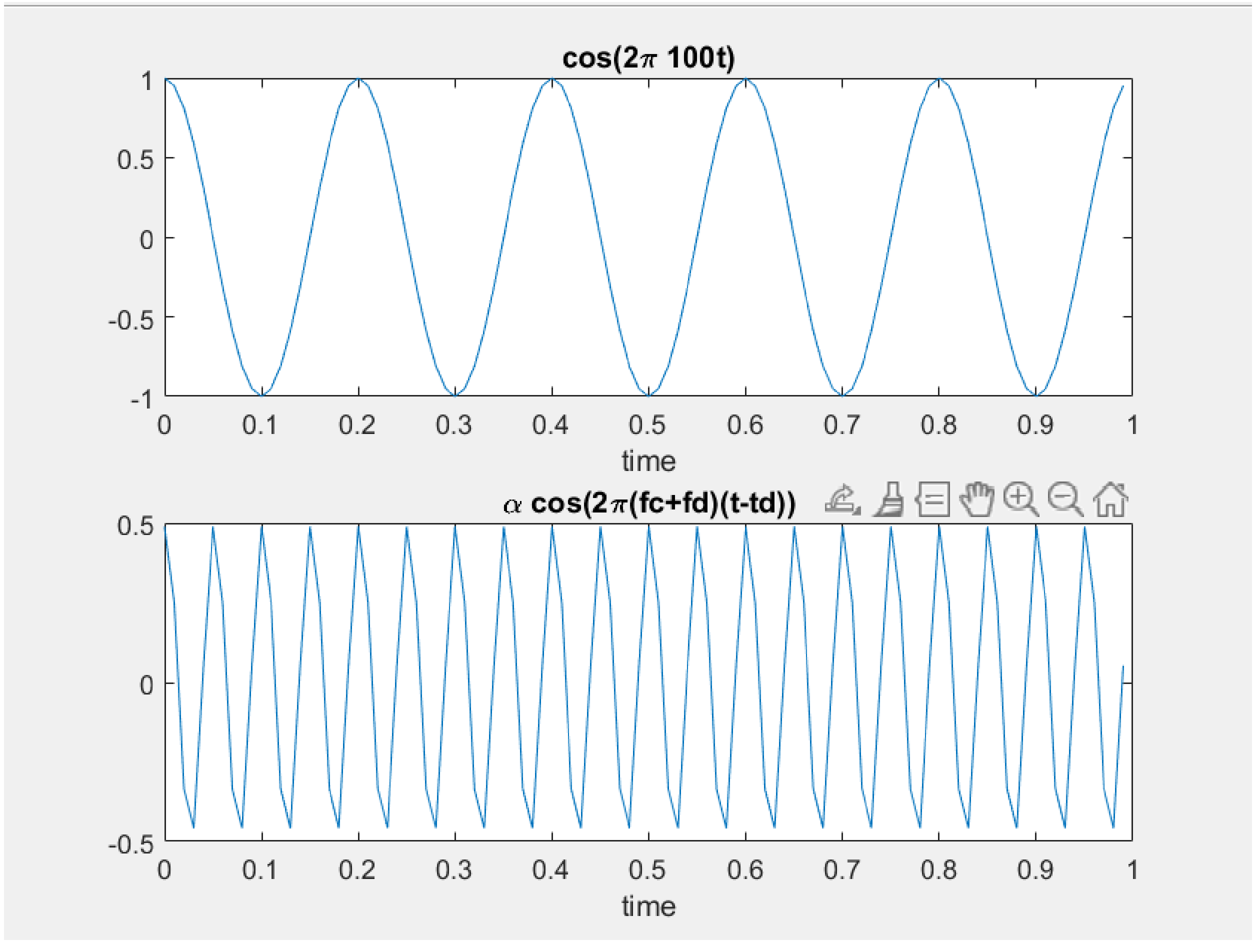
Narges Sadat Seyed Haeri – 810100165

Signals & Systems
CA6 Report

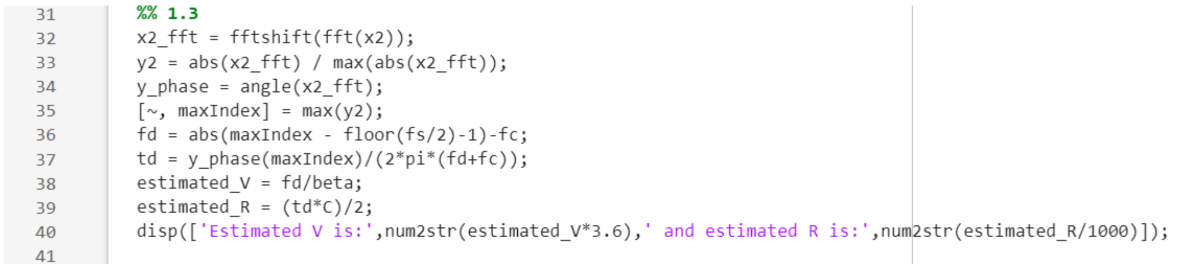
- ***Part 1:***

1.1-2) This code generates plots for both the transmitted and received signals using the specified formulas.

```
%% 1.1
fc = 5;
fs = 100;
t_start = 0;
t_end = 1;
ts = 1 / fs;
t = t_start:ts:t_end - ts;
N = length(t);
f = (-fs / 2):(fs / N):(fs / 2 - fs / N);
x1 = cos(2 * pi * fc * t);
figure
subplot(2, 1, 1)
plot(t, x1)
title('cos(2\pi 100t)')
xlabel('time')

%% 1.2
beta = 0.3;
V = 50;
C = 3e8;
R = 250000;
alpha = 0.5;
td = (2 * R) / C;
fd = beta * V;
x2 = alpha * cos(2 * pi * (fc+fd) * (t-td));
subplot(2, 1, 2)
plot(t, x2)
title('\alpha cos(2\pi(fc+fd)(t-td))')
```



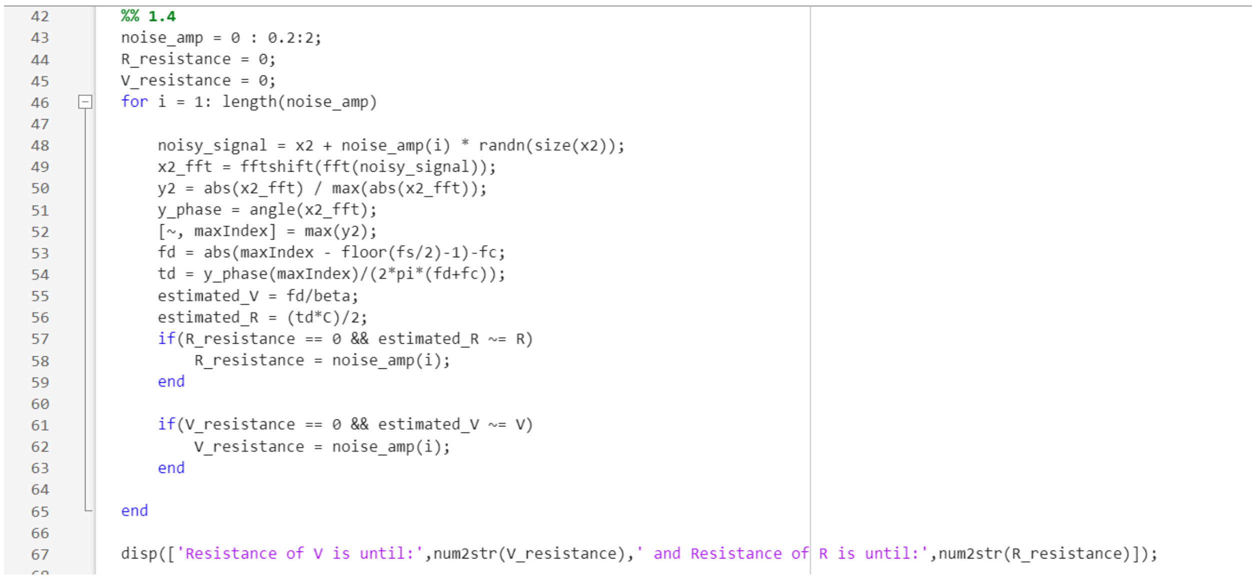
1.3) This code performs signal processing operations on the received signal by applying the Fast Fourier Transform (FFT) and extracting relevant information. The FFT of the received signal is shifted to center the frequency components, and the magnitude and phase information are computed. The dominant frequency component and its corresponding index are identified, allowing the calculation of frequency deviation (fd) and time deviation (td). The code then utilizes these deviations, along with predefined constants such as the sampling frequency (fs), carrier frequency (fc), and modulation index (beta), to estimate the velocity (V) and range (R) parameters of a radar system. The results are displayed, showing the estimated velocity in kilometers per hour and the estimated range in kilometers.



```
%% 1.3
x2_fft = fftshift(fft(x2));
y2 = abs(x2_fft) / max(abs(x2_fft));
y_phase = angle(x2_fft);
[~, maxIndex] = max(y2);
fd = abs(maxIndex - floor(fs/2)-1)-fc;
td = y_phase(maxIndex)/(2*pi*(fd+fc));
estimated_V = fd/beta;
estimated_R = (td*C)/2;
disp(['Estimated V is:',num2str(estimated_V*3.6),' and estimated R is:',num2str(estimated_R/1000)]);
```

```
Estimated V is:180 and estimated R is:250
```

1.4) In this code, we evaluate the resistance of a radar system to noise by iteratively introducing varying levels of noise to the received signal. For each noise amplitude in the range from 0 to 2 with a step of 0.2, we added Gaussian noise to the original signal and analyzed the resulting signal through FFT and signal processing steps. The code calculates the estimated velocity (V) and range (R) parameters based on the noisy signal and compares them with the true values (V and R). The resistance of the system to noise is determined by identifying the maximum noise amplitude at which the estimated V and R deviate from their true values. The results are then displayed, showing the resistance threshold for both velocity (V_resistance) and range (R_resistance) in terms of noise amplitude.

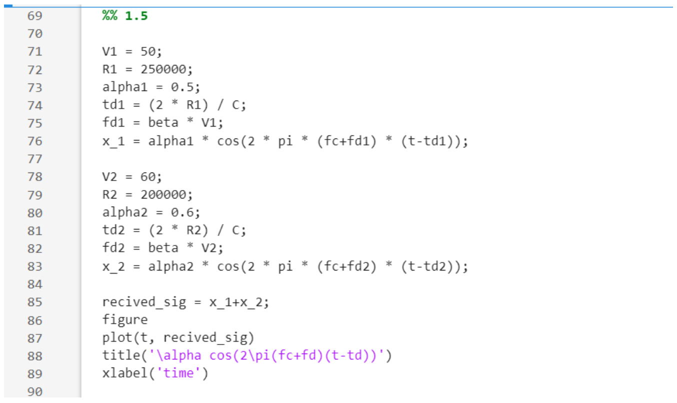


```
%% 1.4
noise_amp = 0 : 0.2:2;
R_resistance = 0;
V_resistance = 0;
for i = 1: length(noise_amp)

    noisy_signal = x2 + noise_amp(i) * randn(size(x2));
    x2_fft = fftshift(fft(noisy_signal));
    y2 = abs(x2_fft) / max(abs(x2_fft));
    y_phase = angle(x2_fft);
    [~, maxIndex] = max(y2);
    fd = abs(maxIndex - floor(fs/2)-1)-fc;
    td = y_phase(maxIndex)/(2*pi*(fd+fc));
    estimated_V = fd/beta;
    estimated_R = (td*C)/2;
    if(R_resistance == 0 && estimated_R ~= R)
        R_resistance = noise_amp(i);
    end

    if(V_resistance == 0 && estimated_V ~= V)
        V_resistance = noise_amp(i);
    end

end

disp(['Resistance of V is until:',num2str(V_resistance),' and Resistance of R is until:',num2str(R_resistance)]);
```

The outcome for the provided received signal is as follows:

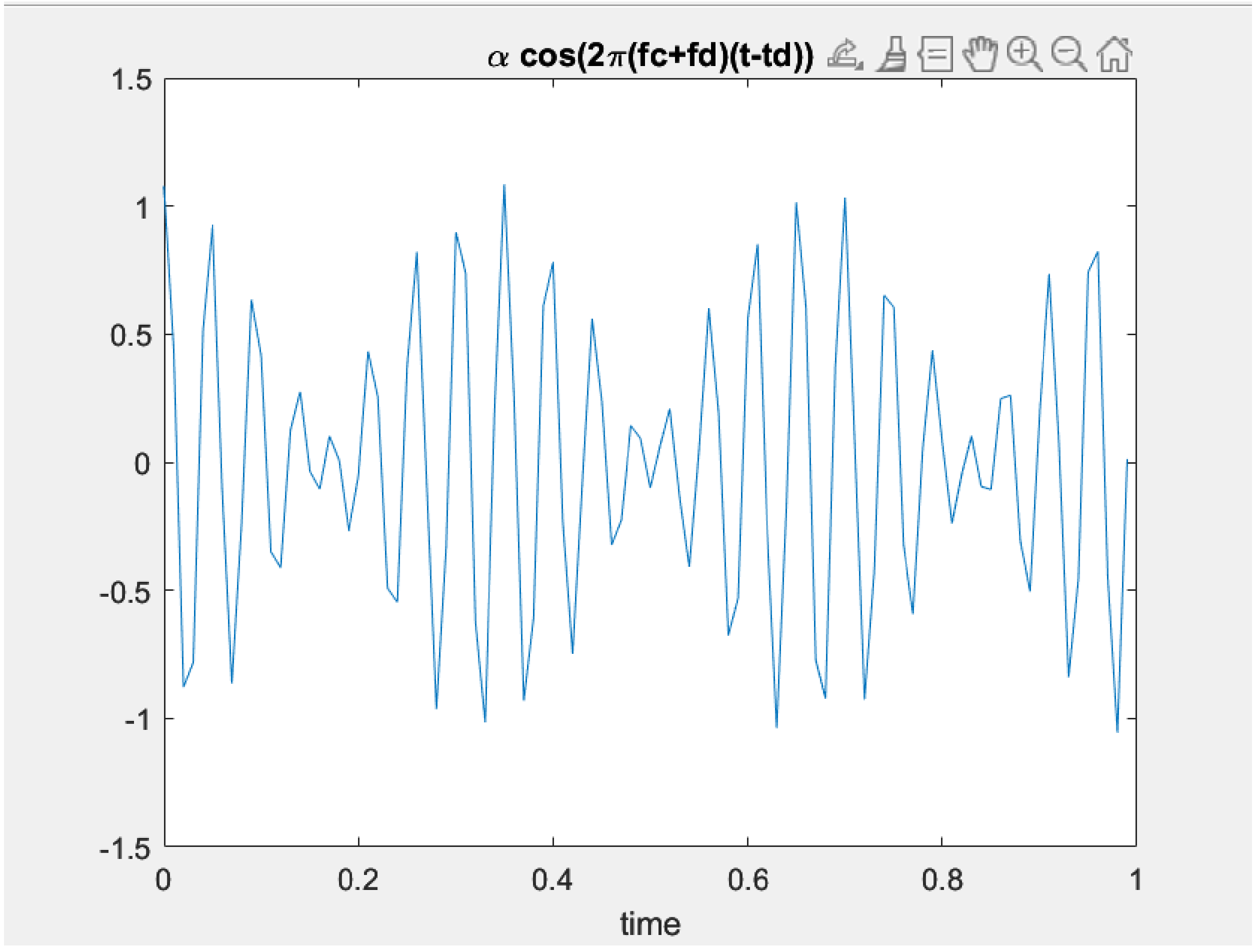
```
Resistance of V is until:1 and Resistance of R is until:0.2
```

1.5) This code generates plot for received signals using the specified formulas.



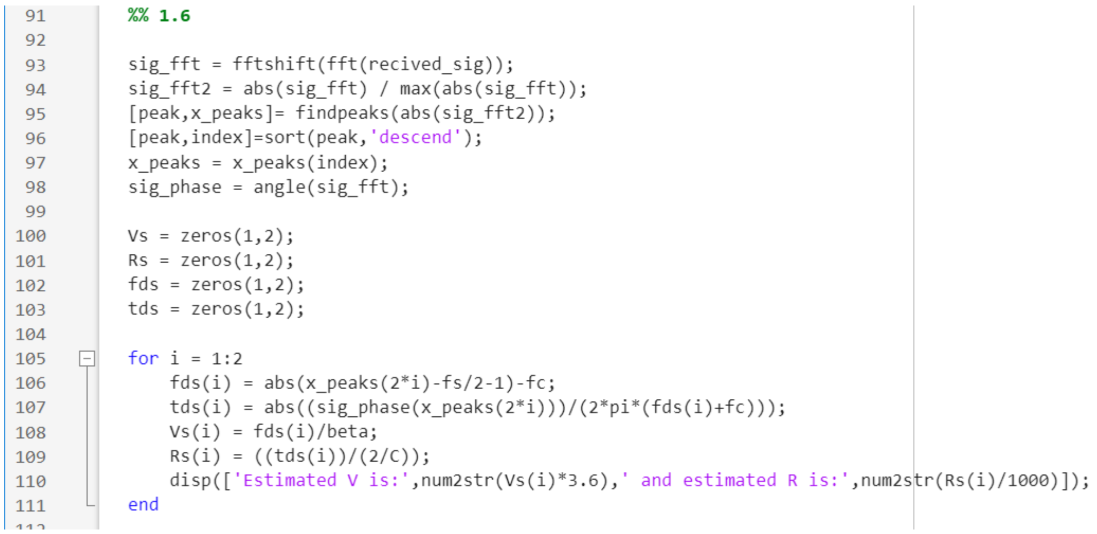
```
%% 1.5

V1 = 50;
R1 = 250000;
alpha1 = 0.5;
td1 = (2 * R1) / C;
fd1 = beta * V1;
x_1 = alpha1 * cos(2 * pi * (fc+fd1) * (t-td1));

V2 = 60;
R2 = 200000;
alpha2 = 0.6;
td2 = (2 * R2) / C;
fd2 = beta * V2;
x_2 = alpha2 * cos(2 * pi * (fc+fd2) * (t-td2));

recived_sig = x_1+x_2;
figure
plot(t, recived_sig)
title('\alpha cos(2\pi(fc+fd)(t-td))')
xlabel('time')
```

Here is the received signal:



1.6) This code processes a received signal through Fast Fourier Transform (FFT) operations to analyze its frequency components and extract relevant information. After obtaining the magnitude and phase of the FFT-shifted signal, the code identifies peaks in the spectrum using the 'findpeaks' function and sorts them in descending order. Subsequently, it calculates the frequency deviation (fds), time deviation (tds), estimated velocity (Vs), and estimated range (Rs) for the top two peaks. The results, representing the estimated velocity in kilometers per hour and estimated range in kilometers, are then displayed.

```
Estimated V is:216 and estimated R is:200
Estimated V is:180 and estimated R is:250
```



```
%% 1.6

sig_fft = fftshift(fft(recived_sig));
sig_fft2 = abs(sig_fft) / max(abs(sig_fft));
[peak,x_peaks]= findpeaks(abs(sig_fft2));
[peak,index]=sort(peak,'descend');
x_peaks = x_peaks(index);
sig_phase = angle(sig_fft);

Vs = zeros(1,2);
Rs = zeros(1,2);
fds = zeros(1,2);
tds = zeros(1,2);

for i = 1:2
    fds(i) = abs(x_peaks(2*i)-fs/2-1)-fc;
    tds(i) = abs((sig_phase(x_peaks(2*i)))/(2*pi*(fds(i)+fc)));
    Vs(i) = fds(i)/beta;
    Rs(i) = ((tds(i))/(2/C));
    disp(['Estimated V is:',num2str(Vs(i)*3.6),' and estimated R is:',num2str(Rs(i)/1000)]);
end
```

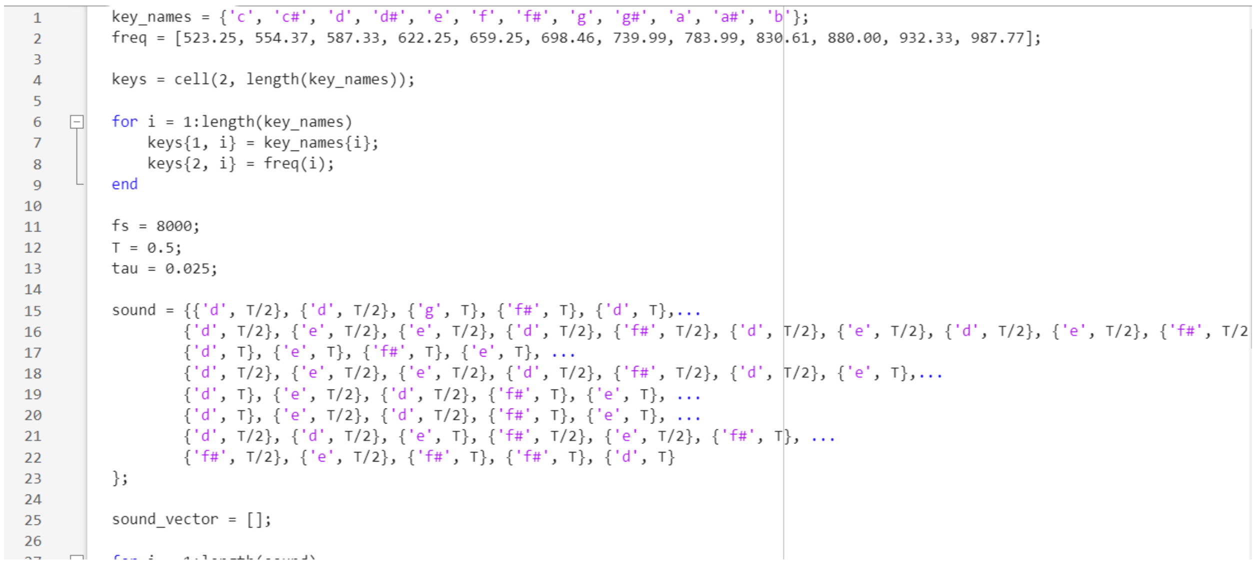
1.7) No, it may not be possible to distinguish between two objects if their speeds are equal because their frequencies (Vs) in the received signal would overlap in the FFT transform. In radar signal processing, the ability to resolve and distinguish between targets depends on the frequency resolution, and when the speeds are identical, the frequencies become indistinguishable in the FFT output. The minimum speed difference required for accurate estimation corresponds to the resolution of the frequency analysis.

1.8) Yes, it is possible to distinguish between two objects even if their distances (R). The code utilizes the peaks in the FFT transform to extract frequency shift (fd) and time delay (td) information. Since these parameters are primarily determined by the frequencies, variations in distance (R) do not significantly impact the ability to distinguish and estimate the distance and the speed of the objects.

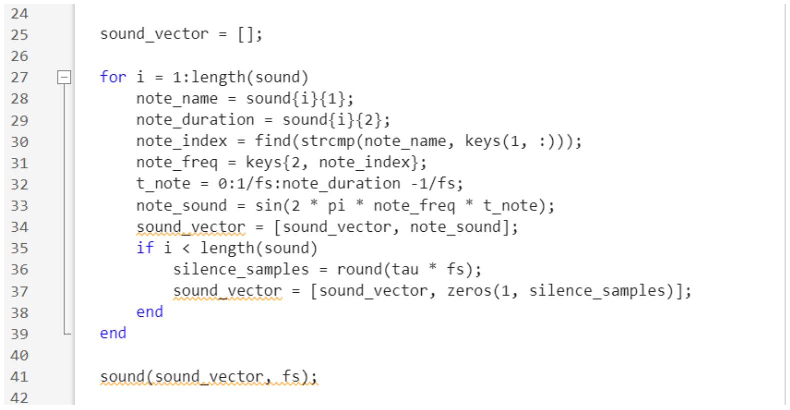
1.9) The approach elucidated in section 1.5 can be employed for more objects.

- ***Part2:***

2.1) This code generates a musical sequence by combining individual notes specified in the sound cell array, each represented by a note name (e.g., 'd') and duration. The code then converts this musical sequence into a time-domain audio signal (sound_vector) by mapping each note to its corresponding frequency and generating a sine wave for the specified duration. A short pause, or silence, is introduced between consecutive notes to enhance the musical rendition. The frequencies and durations of the notes are predefined in the keys cell array, and the resulting audio signal is played using the sound function with a given sampling rate (fs).



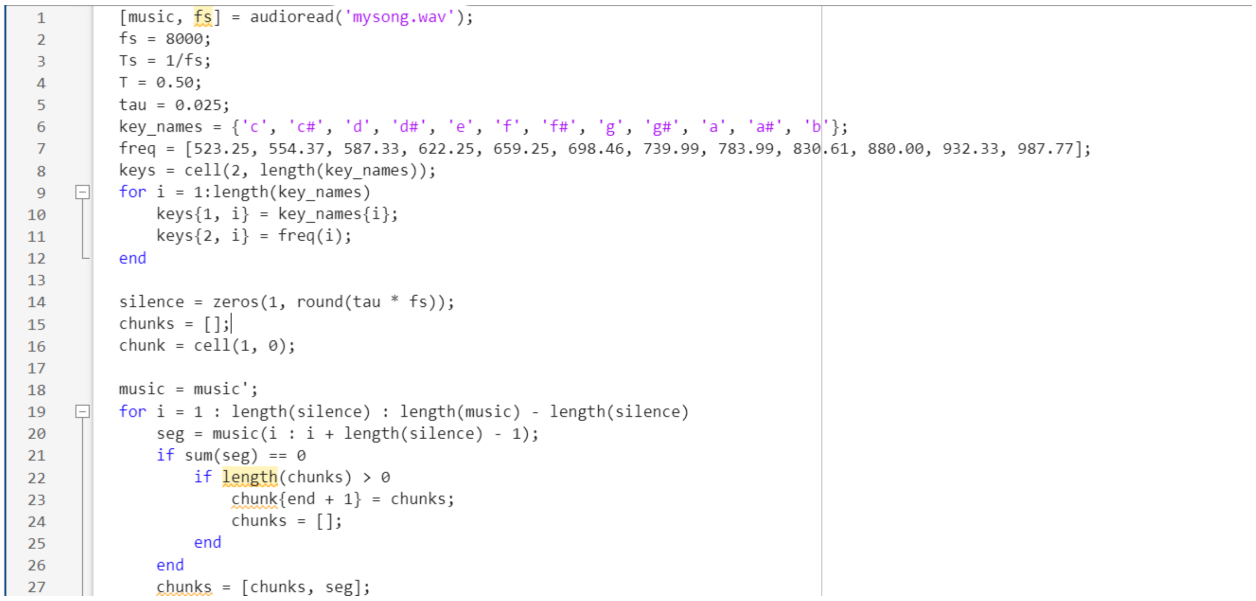
```
key_names = {'c', 'c#', 'd', 'd#', 'e', 'f', 'f#', 'g', 'g#', 'a', 'a#', 'b'};
freq = [523.25, 554.37, 587.33, 622.25, 659.25, 698.46, 739.99, 783.99, 830.61, 880.00, 932.33, 987.77];

keys = cell(2, length(key_names));

for i = 1:length(key_names)
    keys{1, i} = key_names{i};
    keys{2, i} = freq(i);
end

fs = 8000;
T = 0.5;
tau = 0.025;

sound = {{'d', T/2}, {'d', T/2}, {'g', T}, {'f#', T}, {'d', T},...
        {'d', T/2}, {'e', T/2}, {'e', T/2}, {'d', T/2}, {'f#', T/2}, {'d', T/2}, {'e', T/2}, {'d', T/2}, {'e', T/2}, {'f#', T/2
        {'d', T}, {'e', T}, {'f#', T}, {'e', T}, ...
        {'d', T/2}, {'e', T/2}, {'e', T/2}, {'d', T/2}, {'f#', T/2}, {'d', T/2}, {'e', T},...
        {'d', T}, {'e', T/2}, {'d', T/2}, {'f#', T}, {'e', T}, ...
        {'d', T}, {'e', T/2}, {'d', T/2}, {'f#', T}, {'e', T}, ...
        {'d', T/2}, {'d', T/2}, {'e', T}, {'f#', T/2}, {'e', T/2}, {'f#', T}, ...
        {'f#', T/2}, {'e', T/2}, {'f#', T}, {'f#', T}, {'d', T}
};

sound_vector = [];
```



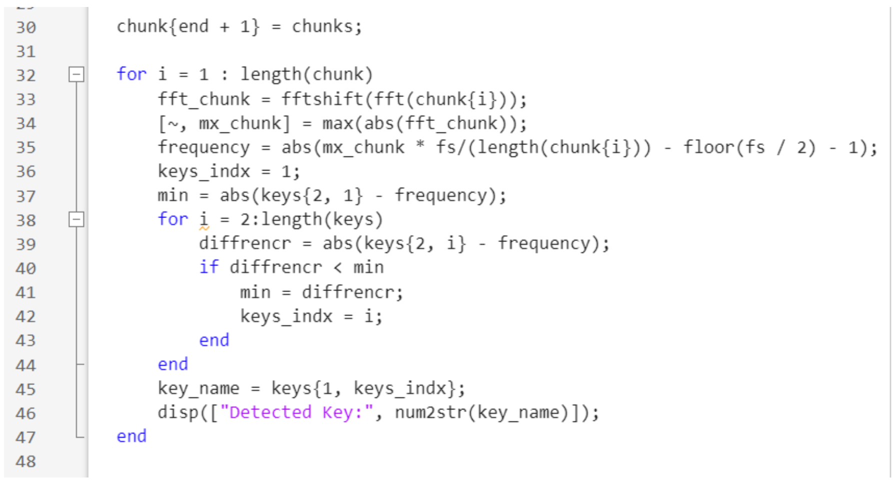
```

sound_vector = [];

for i = 1:length(sound)
    note_name = sound{i}{1};
    note_duration = sound{i}{2};
    note_index = find(strcmp(note_name, keys(1, :)));
    note_freq = keys{2, note_index};
    t_note = 0:1/fs:note_duration -1/fs;
    note_sound = sin(2 * pi * note_freq * t_note);
    sound_vector = [sound_vector, note_sound];
    if i < length(sound)
        silence_samples = round(tau * fs);
        sound_vector = [sound_vector, zeros(1, silence_samples)];
    end
end

sound(sound_vector, fs);

```

2.2) the code is the same as the previous part but instead of using sound function we use audiowrite to save the music as 'mysong.wav'. length half note is 2000 and full note is 4000 and the silence note is 200.

2.3) In this code, we start by loading an audio file, 'mysong.wav,' and configuring parameters such as the sampling rate (fs), duration of silence (tau), and time step (Ts). We then process the audio by segmenting it into chunks based on periods of silence. Each chunk is subjected to analysis through the Fast Fourier Transform (FFT), revealing its frequency spectrum. By identifying the peak in the FFT, we determine the dominant frequency in each chunk. Next, we map this frequency to the nearest musical key using predefined key frequencies stored in the 'keys' cell array. The code iterates through the chunks, displaying the detected musical key for each, providing insights into the changing harmonic content of the audio.



```
[music, fs] = audioread('mysong.wav');
fs = 8000;
Ts = 1/fs;
T = 0.50;
tau = 0.025;
key_names = {'c', 'c#', 'd', 'd#', 'e', 'f', 'f#', 'g', 'g#', 'a', 'a#', 'b'};
freq = [523.25, 554.37, 587.33, 622.25, 659.25, 698.46, 739.99, 783.99, 830.61, 880.00, 932.33, 987.77];
keys = cell(2, length(key_names));
for i = 1:length(key_names)
    keys{1, i} = key_names{i};
    keys{2, i} = freq(i);
end

silence = zeros(1, round(tau * fs));
chunks = [];
chunk = cell(1, 0);

music = music';
for i = 1 : length(silence) : length(music) - length(silence)
    seg = music(i : i + length(silence) - 1);
    if sum(seg) == 0
        if length(chunks) > 0
            chunk{end + 1} = chunks;
            chunks = [];
        end
    end
    chunks = [chunks, seg];
```



```
chunk{end + 1} = chunks;

for i = 1 : length(chunk)
    fft_chunk = fftshift(fft(chunk{i}));
    [~, mx_chunk] = max(abs(fft_chunk));
    frequency = abs(mx_chunk * fs/(length(chunk{i})) - floor(fs / 2) - 1);
    keys_indx = 1;
    min = abs(keys{2, 1} - frequency);
    for i = 2:length(keys)
        diffrencr = abs(keys{2, i} - frequency);
        if diffrencr < min
            min = diffrencr;
            keys_indx = i;
        end
    end
    key_name = keys{1, keys_indx};
    disp(["Detected Key:", num2str(key_name)]);
end
```