LBYDISP Laboratory Report

Name: **Group 7** Section: **S11B**

Alamay, Carl Justine S. (Lab Report, Explanations, Testing, and Coding)

Pimental, Ethan Paolo C. (Finalizations, Explanations, and Code Debugging)

Laboratory Activity Title: “AM-FM Sinusoidal Signals”

Files Used: https://drive.matlab.com/sharing/1bb073d1-2b98-4b11-8f87-1b6e25b3b53f

Instructions: , for each of the sections indicated, please provide your corresponding answers to each of the questions indicated in the lab activity section. Please provide relevant information, i.e. **source code, graphs, explanations/answers to the question/s.**

2.4 MATLAB Synthesis of Chirp Signals

```
fsamp = 11025;
dt = 1/fsamp;
dur = 1.8;
tt = 0 : dt : dur;
psi = 2*pi*(100 + 200*tt + 500*tt.*tt);
xx = real( 7.7*exp(j*psi) );
soundsc( xx, fsamp );
```

(a) The total duration of the synthesized signal is 1.8 seconds given by the dur variable. The number of samples, which can be determined by getting the length of the tt variable, is 19846. It can be determined using the code snippet:

```
numSamples = length(tt);
```

(b) **x(tn) = A cos(2 * pi * u * tn^2 + 2 * pi * f0 * tn + phi).**

- The value for tn is given by the time vector, tt, which is 0 to 1.8 with an interval if 1/11025.
- A is the amplitude given by the xx variable, which is 7.7.
- U is the coefficient in the phase, which is 500.
- f0 is the time-varying frequency component, which is 100.
- Phi is the phase shift of the signal, which is 0.

(c)
```
figure;
plot(psi);
```
By plotting the psi variable, where the equation of the signal is defined, we can determine that the minimum frequency is 628.319 Hz and the maximum frequency is 13069 Hz.

3.2 Function of a Chirp

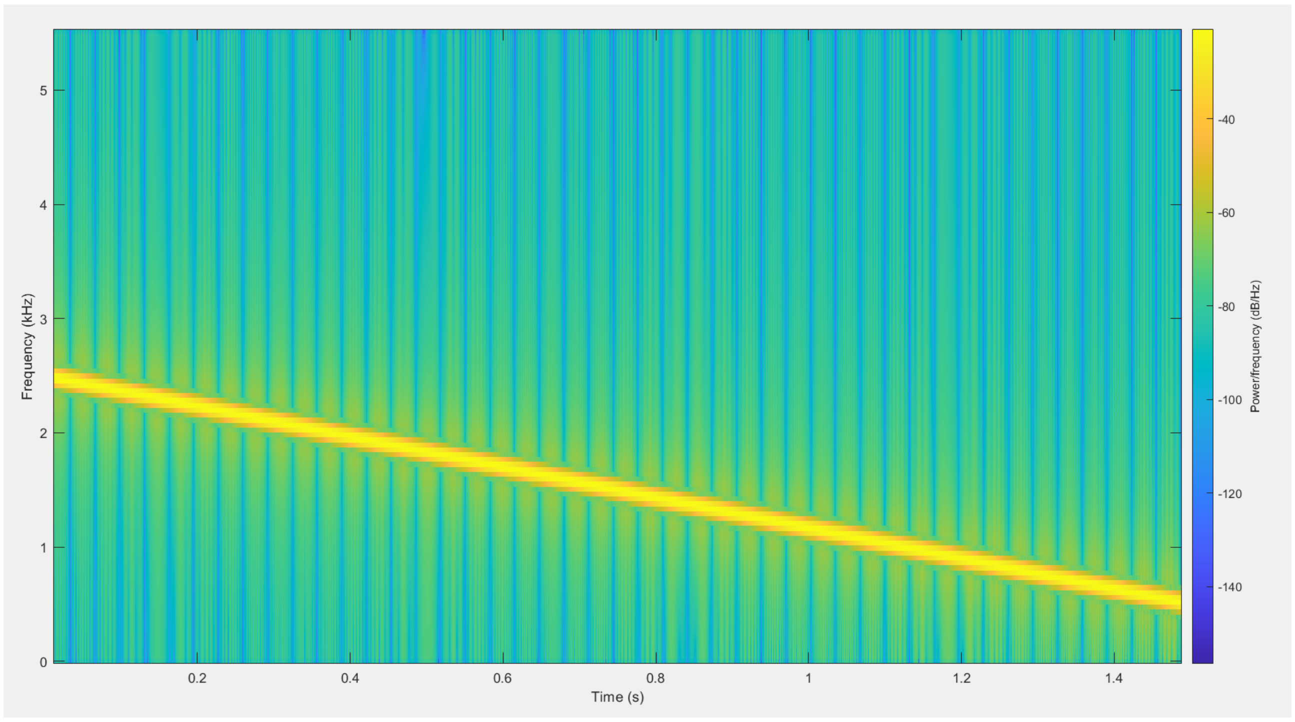
```
function [xx, tt] = mychirp(f1, f2, dur, fsamp)

if nargin < 4, fsamp = 11025, end;

tt = 0 : 1/fsamp : dur;
k = (f2 - f1) / dur;
phi = 2 * pi * (f1 * tt + 0.5 * k * tt.^2);

xx =sin(phi);
end
```

```
f1 = 2500;
f2 = 500;
dur = 1.5;

[xx, tt] = mychirp(f1, f2, dur, fsamp);

soundsc(xx, fsamp);
spectrogram(xx, 256, 250, [], fsamp, 'yaxis');

figure;
plot(xx);
```

Code for the mychirp function, and initial call parameters.

The code above illustrates the implementation of the mychirp function, and the chosen parameters specified in the laboratory manual. It starts off with a frequency of 2500Hz, which is set to the variable f1, and smoothly transitions to 500HZ, which is the ending frequency set to the variable f2, over the span of 1.5 seconds.

After the implementation of the function, the mychirp file takes in the aforementioned variables and creates the sound of a chirp which corresponds to the time vector stored in tt. The result of the function call is then played using the soundsc function, which is then illustrated and visualized through a spectrogram.



Spectrogram of the initial parameters after running through mychirp (done in lab).

4.1 Beat Notes

```
function [xx, tt] = beat(A, B, fc, delf, fsamp, dur)

A = abs(A);
B = abs(B);

f1 = fc - delf / 2;
f2 = fc + delf / 2;

[sin1, ~] = syn_sin(f1, A, fsamp, dur);
[sin2, tt] = syn_sin(f2, B, fsamp, dur);

xx = sin1 + sin2;
end
```

Function definition of beat.

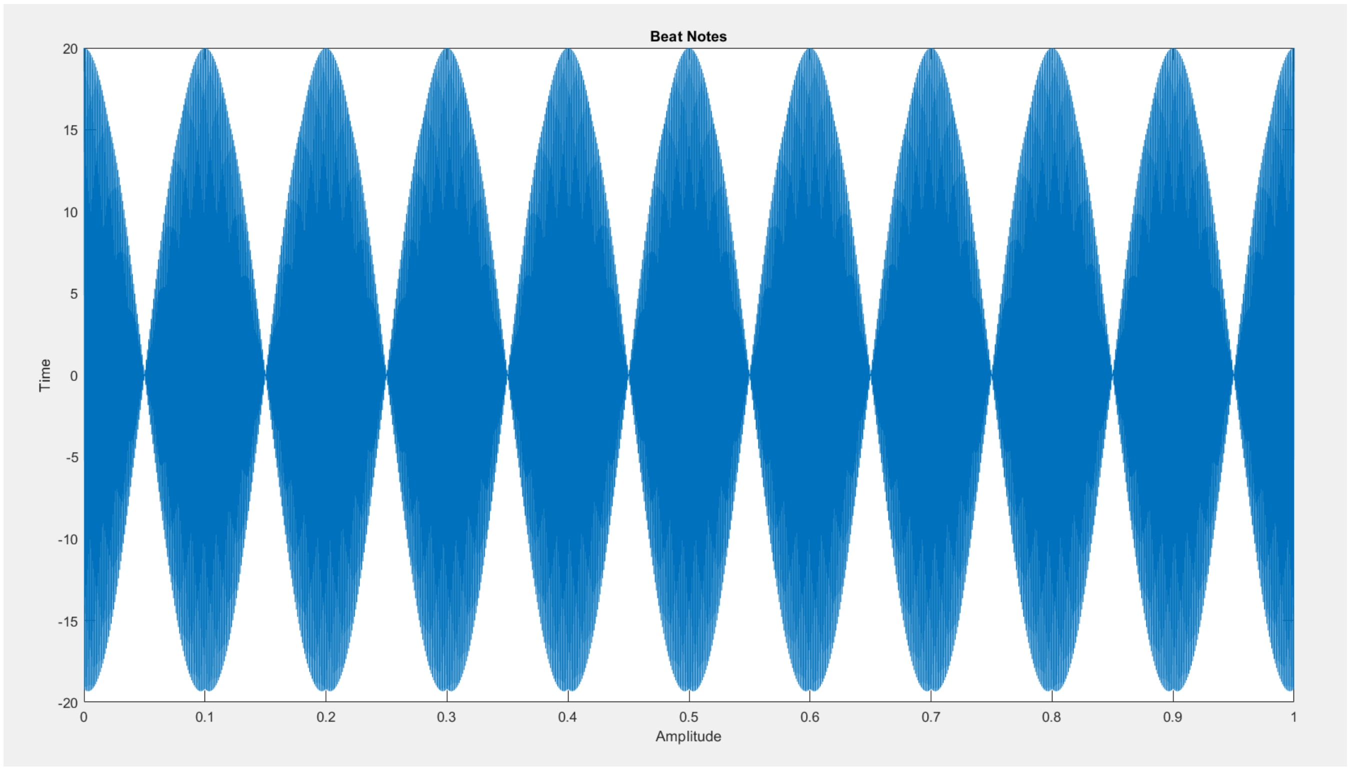
The code defines a function called "beat" that takes in several input parameters: A, B, fc, delf, fsamp, and dur. It returns two outputs, xx and tt. The purpose of this function is to generate a beat signal by combining two cosine signals with slightly different frequencies. The function signature declares the input and output variables:
`[xx, tt] = beat(A, B, fc, delf, fsamp, dur)`

Input Parameters:
A: Amplitude of the first sinusoidal signal.
B: Amplitude of the second sinusoidal signal.
fc: Center frequency around which the beat occurs.
delf: Frequency difference between the two sinusoidal signals, creating the beat effect.
fsamp: Sampling frequency for generating the signals.
dur: Duration of the generated signals.

The beat function utilizes a previously created function called syn_sin to generate sinusoidal signals with the specified frequency, amplitude, sampling frequency, and duration, where 2 sinusoidal signals are created. The two signals are then added together to simulate the beat.

```
A = 10;
B = 10;
fc = 1000;
delf = 10;
fsamp = 11025;
dur = 1;

[xx, tt] = beat(A, B, fc, delf, fsamp, dur);

plot(tt,xx)
xlabel('Amplitude')
ylabel('Time')
title('Beat Notes')
```

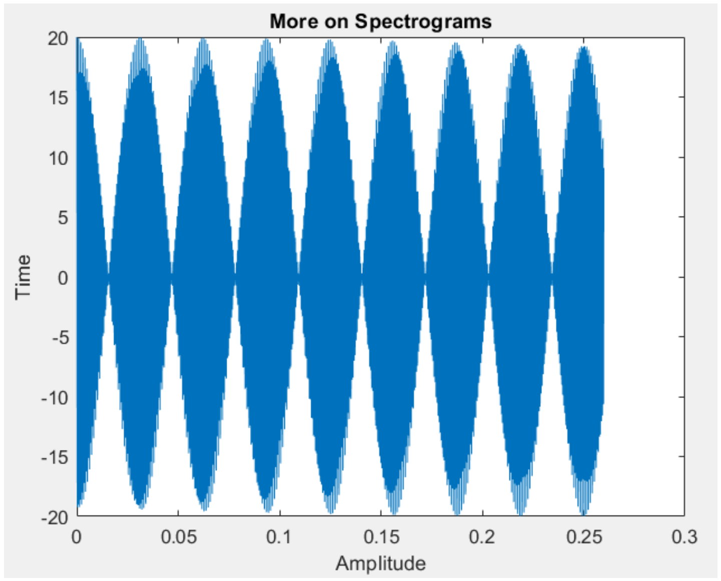
Initial call of the beat function.



Graph of the initial call of the beat function.

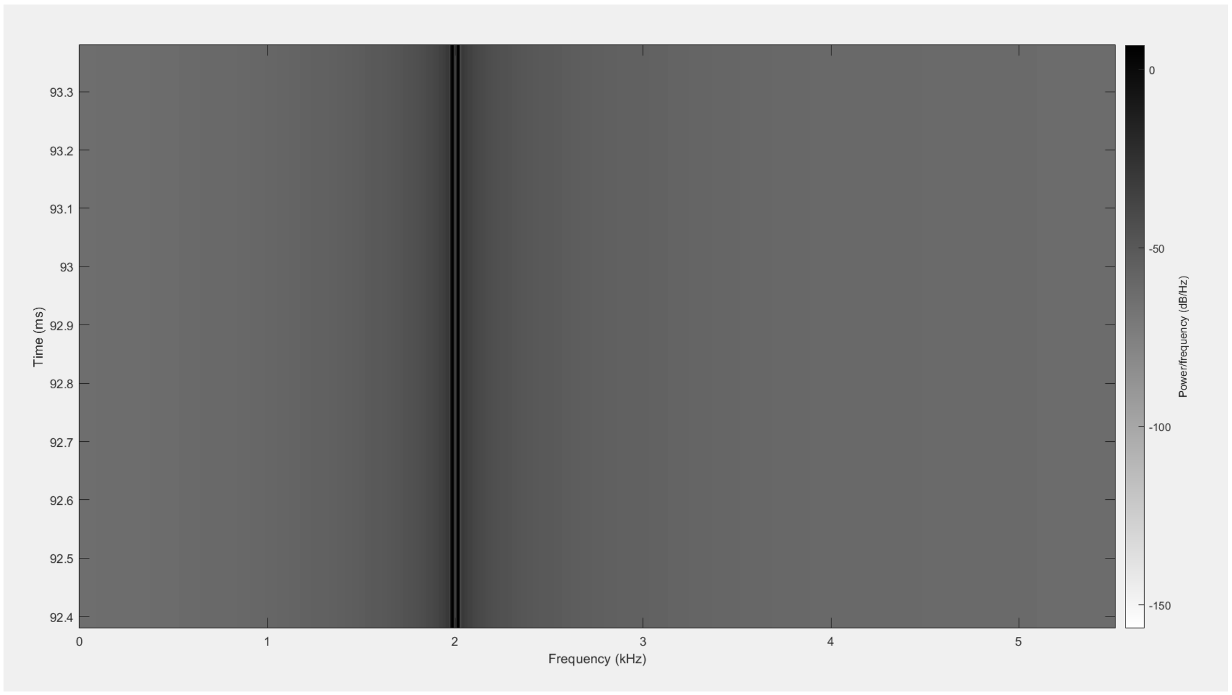
4.2 More on Spectrograms
(a)

```
A= 10;
B= 10;
delf = 32;
dur = 0.26;
fsamp = 11025;
fc = 2000;

[xx, tt] = beat(A, B, fc, delf, fsamp, dur);

plot(tt,xx)
xlabel('Amplitude')
ylabel('Time')
title('More on Spectrograms')
```



Function call graph of the specified parameters.

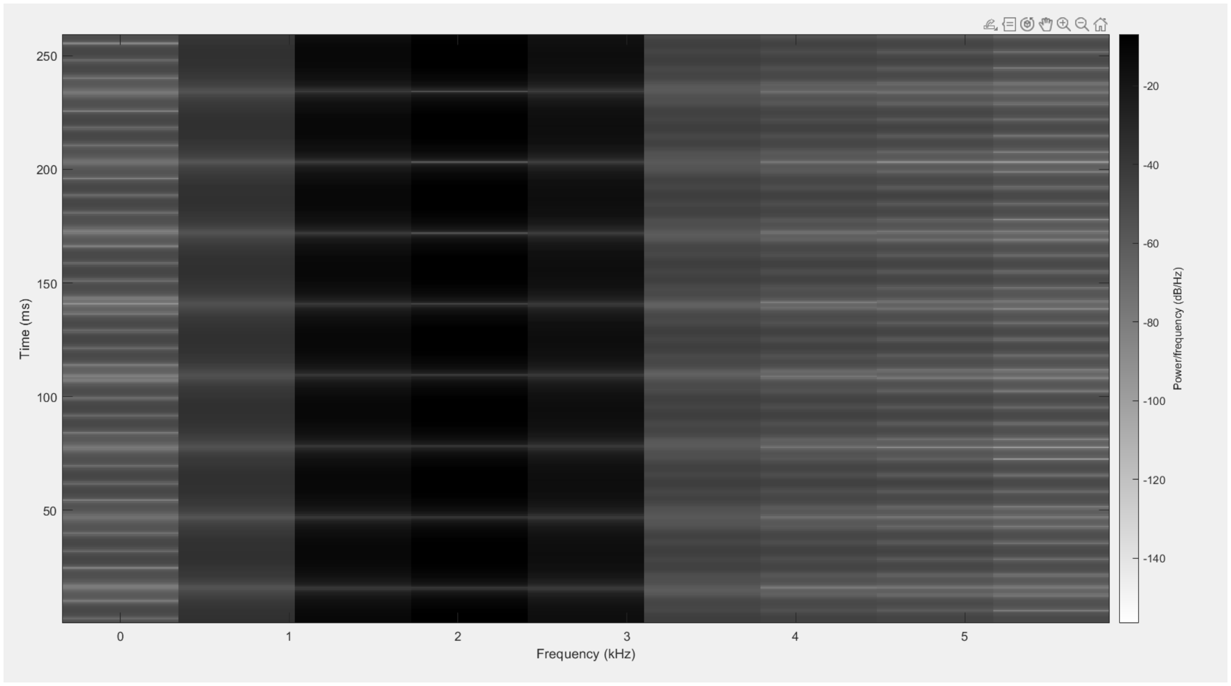
(b)



Spectrogram of the beat signal xx with window length of 2048.

Given the specified parameters, the spectrogram seems to have 2 distinct vertical bands which correspond to the different frequency components used in the signal. Since the specified frequency difference is 32 Hz, it's expected that that the central frequency of 2000 Hz will show modulations. The given frequencies seem to be correct and according to the parameters since the central frequency is directly graphed onto 2 kHz, and the frequency difference of 32 Hz created the distinct lines present in the graph.

(c)



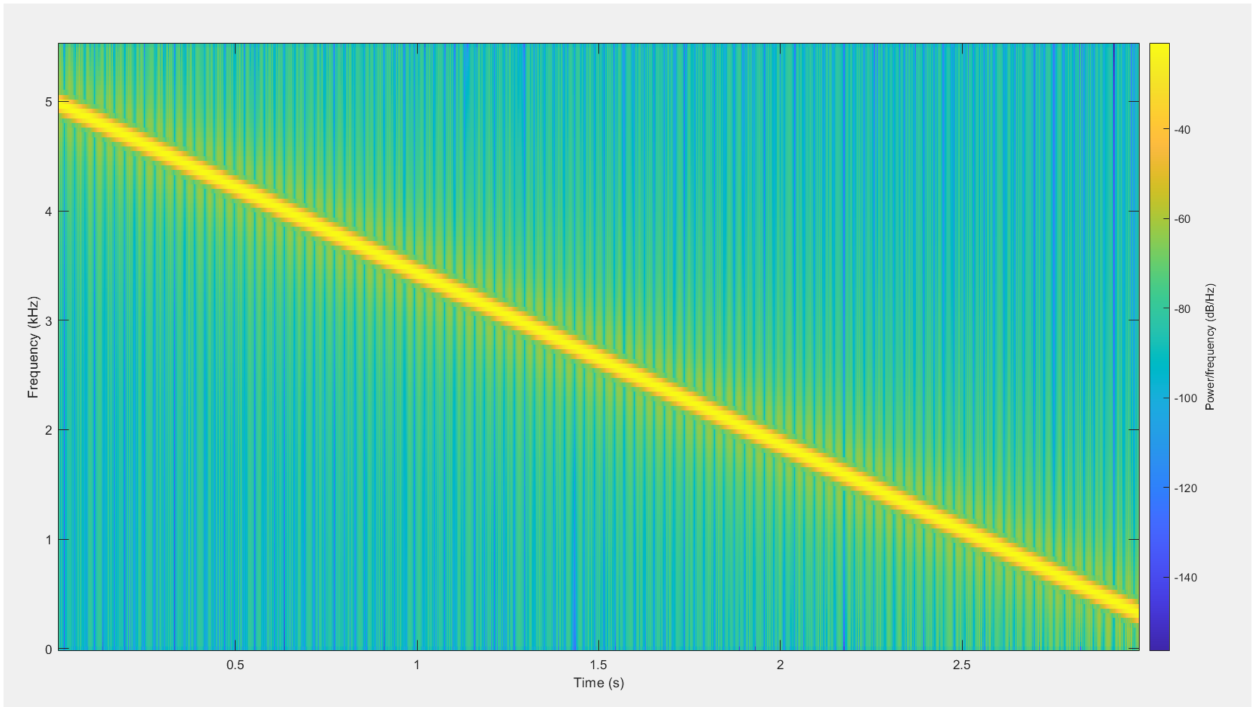
Spectrogram of the beat signal xx with window length of 16.

With the window length being 16, it's expected that the spectrogram is going to have reduced frequency resolution as compared to the previous iteration. The 16 samples per window captures short term changes in the signal, which may not be effective in showing the frequency modulations over certain durations.

Comparing the 2 spectrograms with completely different window lengths, generating a spectrogram with a longer window length may provide better frequency resolutions but sacrifices time resolution. Meanwhile, a spectrogram with a short window length offers subpar frequency resolution but provides better time resolution.

4.3 Spectrogram of a Chirp

```
f1 = 5000;
f2 = 300;
dur = 3;
fsamp = 11025;

[xx, tt] = mychirp(f1, f2, dur, fsamp);
soundsc(xx, fsamp);

spectrogram(xx, 256, 250, [], fsamp, 'yaxis');
```

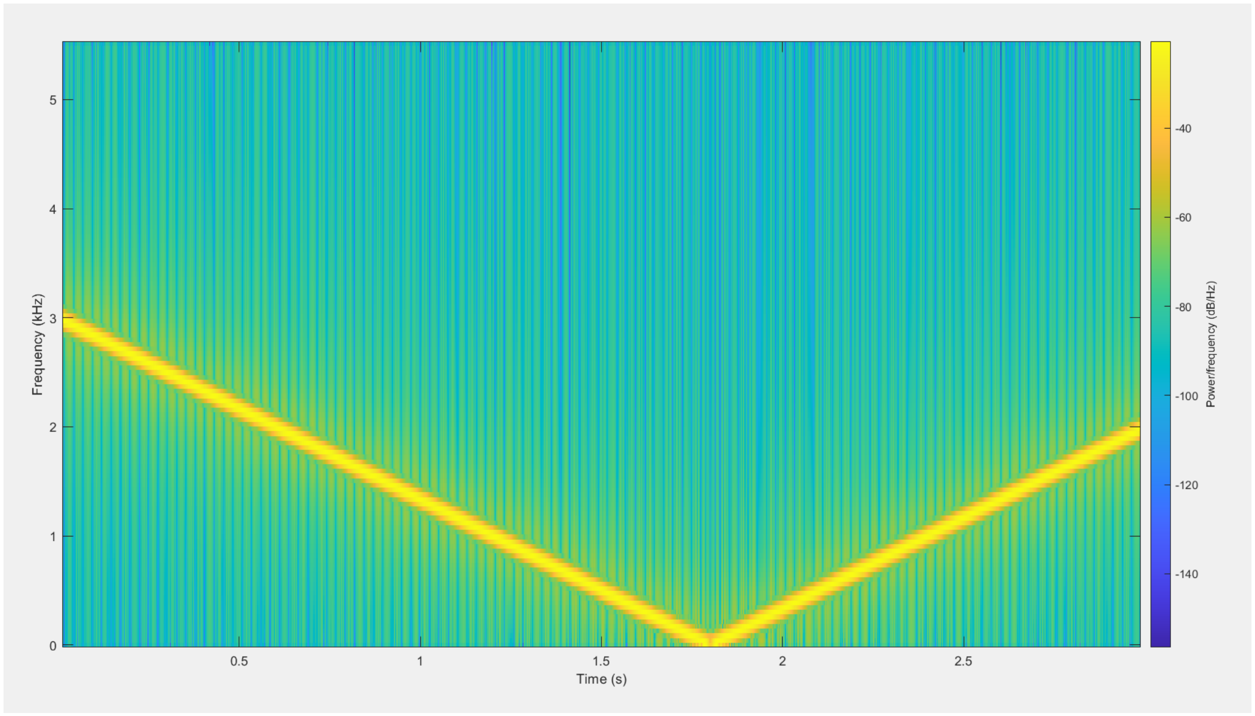


Function call of mychirp with the specified parameters, along with spectrogram.

Upon listening to the generated sound through mychirp, we should be able to observe that it will produce a sound which starts at a high frequency but gradually goes down over the span of 3 seconds. The linearity of the signal can be seen in the spectrogram where the frequency goes in a downward direction. All in all, it chirps down as the spectrogram visually confirms this notion by showing a diagonal line which goes from 5 kHz all the way to 0.3 kHz or 300 Hz over 3 seconds.

4.4 A Chirp Puzzle

```
f1 = 3000;
f2 = -2000;
dur = 3;
fsamp = 11025;

[xx, tt] = mychirp(f1, f2, dur, fsamp);
soundsc(xx, fsamp);

spectrogram(xx, 256, 250, [], fsamp, 'yaxis');
```



Function call of mychirp with the specified parameters, along with spectrogram.

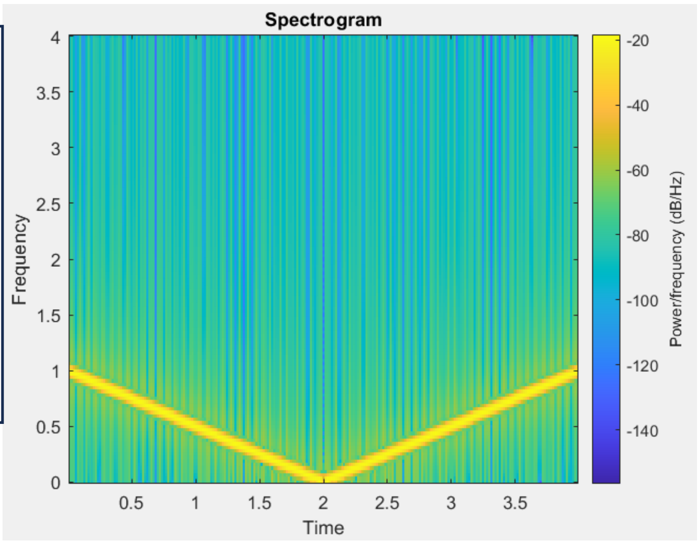
Upon listening to the generated sound, we can observe that the chirp starts at a high frequency then gradually goes down to a negative or lower frequency, which indicates that it chirps both up and down. The changing instantaneous frequency in the second chirp signal implies a movement of frequency components in the spectrum, which also means that the negative frequency indicates that the phase of the signal is either inverted or reversed.

CHALLENGE

```matlab
function [xx, tt] = challchirp(f1, f2, dur, fsamp)

if nargin < 4, fsamp = 11025; end;

t_up = 0 : 1/fsamp : dur/2;
t_down = dur/2 : 1/fsamp : dur;

k_up = (f2 - f1) / (dur/2);
k_down = (f1 - f2) / (dur/2);

phi_up = 2 * pi * (f1 * t_up + 0.5 * k_up * t_up.^2);
phi_down = 2 * pi * (f2 * (t_down - dur/2) + 0.5 * k_down * (t_down - dur/2).^2);

xx_up = sin(phi_up);
xx_down = sin(phi_down);

xx = [xx_up, xx_down];
tt = [t_up, t_down];
end
```

Function definition of chirp used in the challenge.

```matlab
f1 = 1000;
f2 = 0.5;
dur = 4;
fsamp = 8000;
[xx, tt] = challchirp(f1, f2, dur, fsamp);
soundsc(xx, fsamp)

spectrogram(xx, 256, [], [], fsamp, 'yaxis');
title('Spectrogram');
xlabel('Time');
ylabel('Frequency');
```



Function call of the chirp with the given parameters.

The custom chirp function used generates a signal with 2 specified frequencies and divides them into 2 segments based on the time vector, which can simulate the movements of a sine wave. For each segment, it calculates the slope based on the frequency difference and uses it to compute the phase of the chirp signal. The chirp signal exhibits an upward and downward frequency modulation over time, transitioning smoothly from the starting frequency to the ending frequency and back within the specified duration.