Ryerson University

Department of Electrical, Computer, & Biomedical Engineering
Faculty of Engineering & Architectural Science

Course Title:	Signals and Systems II
Course Number:	ELE 632
Semester/Year (e.g.F2016)	W2023

Instructor:	Dr. Androutsos

Assignment/Lab Number:	Lab 5
Assignment/Lab Title:	Sampling and the Discrete Fourier Transform

Submission Date:	2023-04-05
Due Date:	2023-04-09

Student LAST Name	**Student FIRST Name**	**Student Number**	**Section**	**Signature***
Zelenovic	Danilo	501032542	08	DZ

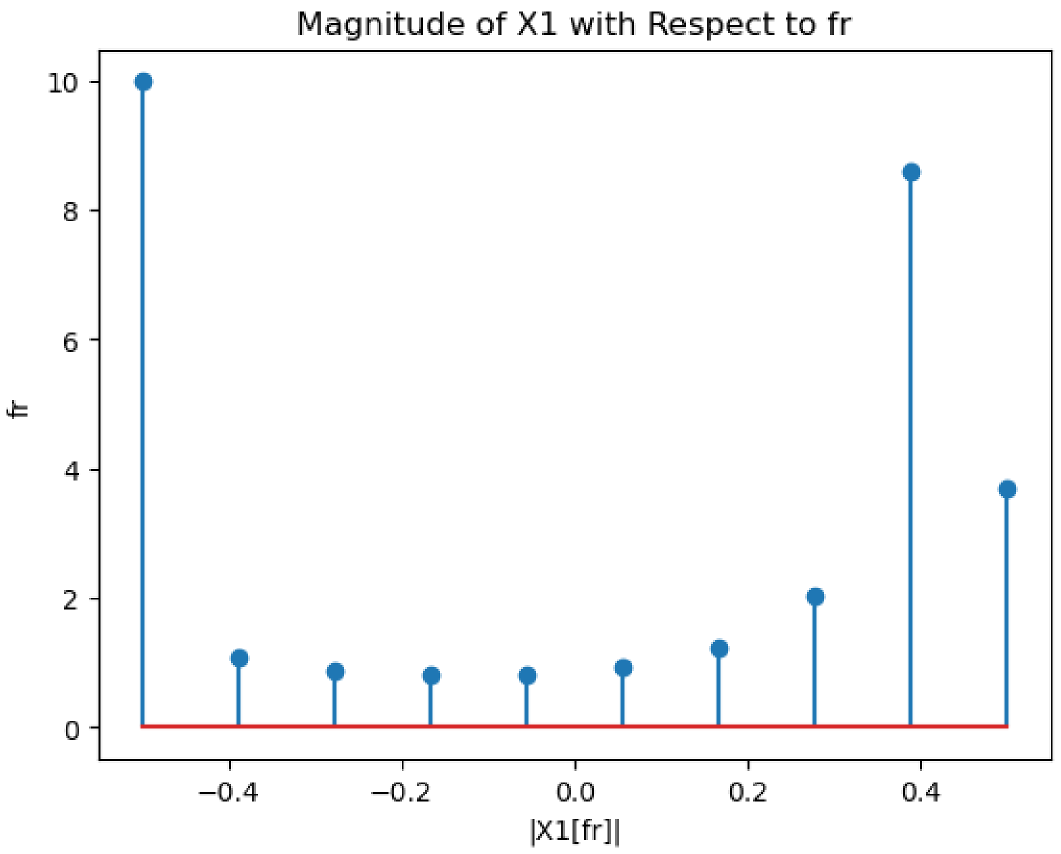
ELE632_lab5_DaniloZelenovic_501032542_Section08

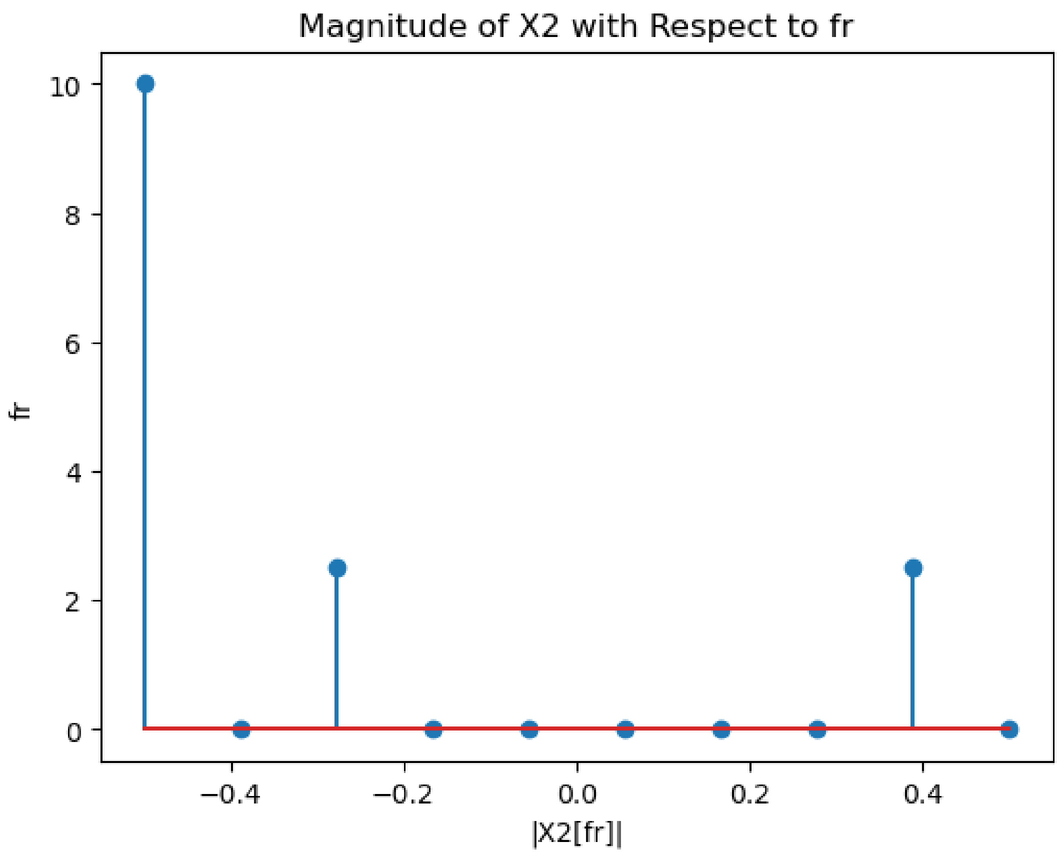
April 5, 2023

Student Number: 5 0 1 0 3 2 5 4 2 a b c d e f g h i

1 —-Part A—-

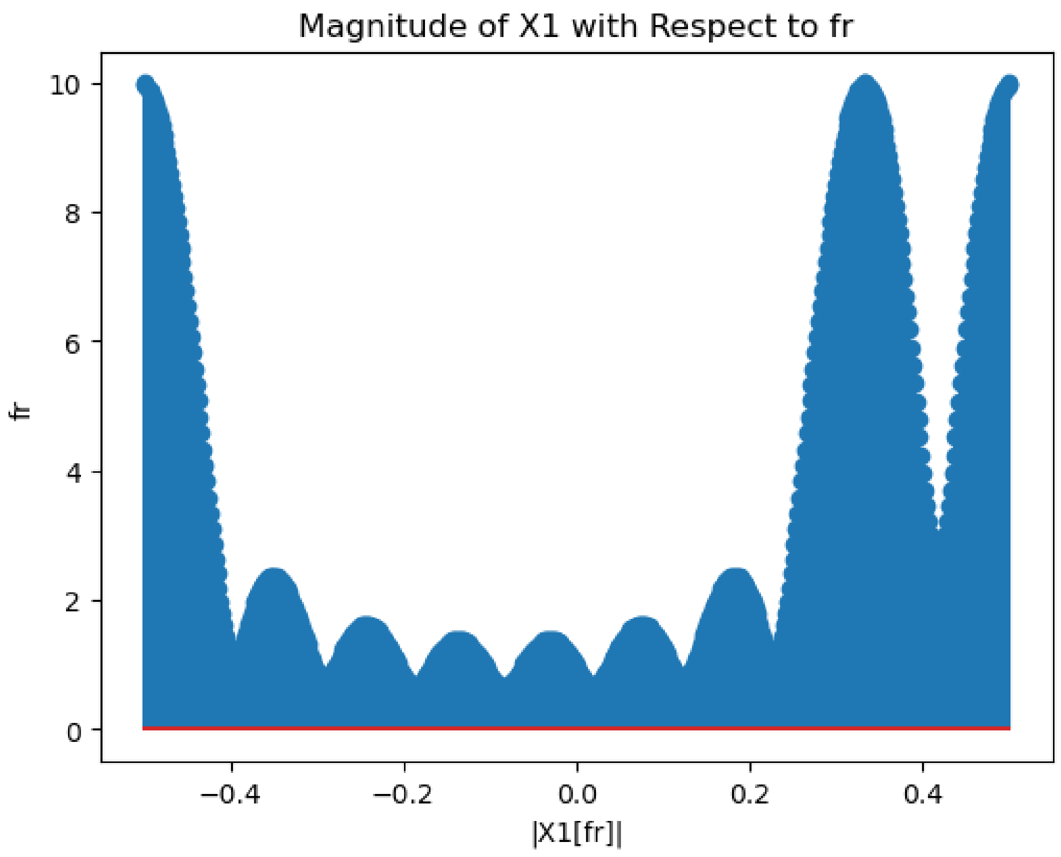
[14]:
```
#A1

import numpy as np
import matplotlib.pyplot as plt

n = np.arange(0, 10)
N = len(n)
fr = np.linspace(-0.5, 0.5, 10)
x1 = np.exp(1j*2*np.pi*((10*5)/100)*n) + np.exp(1j*2*np.pi*(33/100)*n)
x2 = np.cos(2*np.pi*((10*5)/100)*n) + 0.5*np.cos(2*np.pi*((10*3)/100)*n)

X1 = np.fft.fftshift(np.fft.fft(x1))
X2 = np.fft.fftshift(np.fft.fft(x2))

plt.stem(fr, abs(X1))
plt.title("Magnitude of X1 with Respect to fr")
plt.xlabel('|X1[fr]|')
plt.ylabel('fr')
plt.show()

plt.stem(fr, abs(X2))
plt.title("Magnitude of X2 with Respect to fr")
plt.xlabel('|X2[fr]|')
plt.ylabel('fr')
plt.show()
```

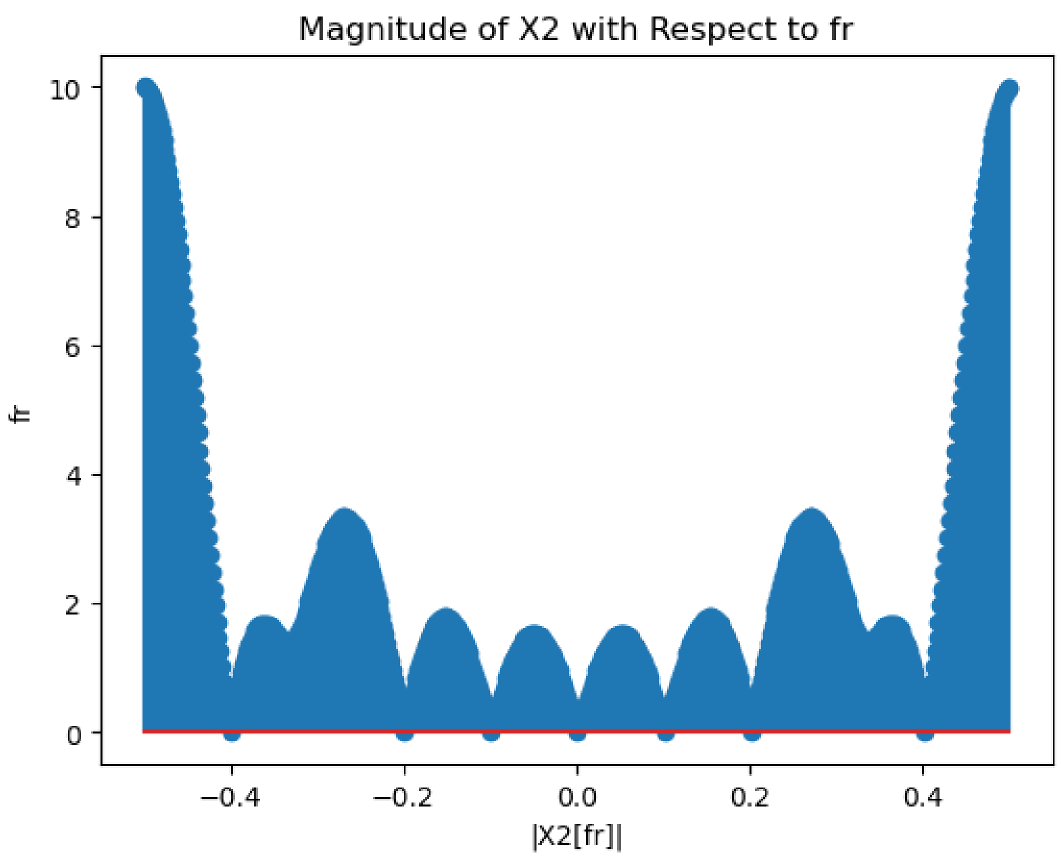

Magnitude of X1 with Respect to fr
fr
|X1[fr]|
10
8
6
4
2
0
−0.4
−0.2
0.0
0.2
0.4



i) X2 seems symmetrical, as X1 clearly appears it cannot be symmetrical. We can see that the sum of X2 is even. ii) Yes it is possible to distinguish, as the plots have different frequency components. iii) We see the other components as it is a linear combination of 2 signals with different frequencies.

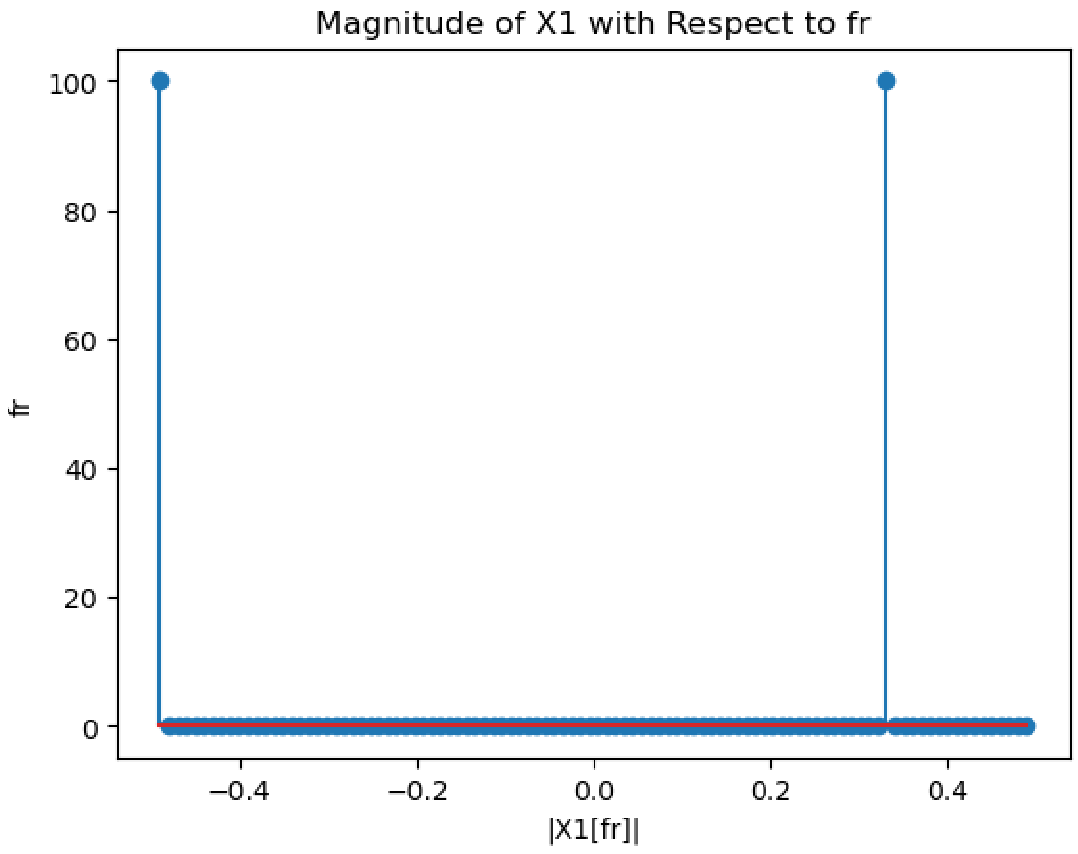
[8]:
```
#A2

import numpy as np
import matplotlib.pyplot as plt

n = np.arange(0, 10)
N = 500
fr = np.linspace(-0.5, 0.5, N)
x1 = np.exp(1j*2*np.pi*((10*5)/100)*n) + np.exp(1j*2*np.pi*(33/100)*n)
x3 = np.concatenate((np.zeros(245), x1, np.zeros(245)))
x2 = np.cos(2*np.pi*((10*5)/100)*n) + 0.5*np.cos(2*np.pi*((10*3)/100)*n)
x4 = np.concatenate((np.zeros(245), x2, np.zeros(245)))

X3 = np.fft.fftshift(np.fft.fft(x3))
X4 = np.fft.fftshift(np.fft.fft(x4))

plt.stem(fr, abs(X3))
plt.title("Magnitude of X1 with Respect to fr")
plt.xlabel('|X1[fr]|')
plt.ylabel('fr')
plt.show()

plt.stem(fr, abs(X4))
plt.title("Magnitude of X2 with Respect to fr")
plt.xlabel('|X2[fr]|')
plt.ylabel('fr')
plt.show()
```

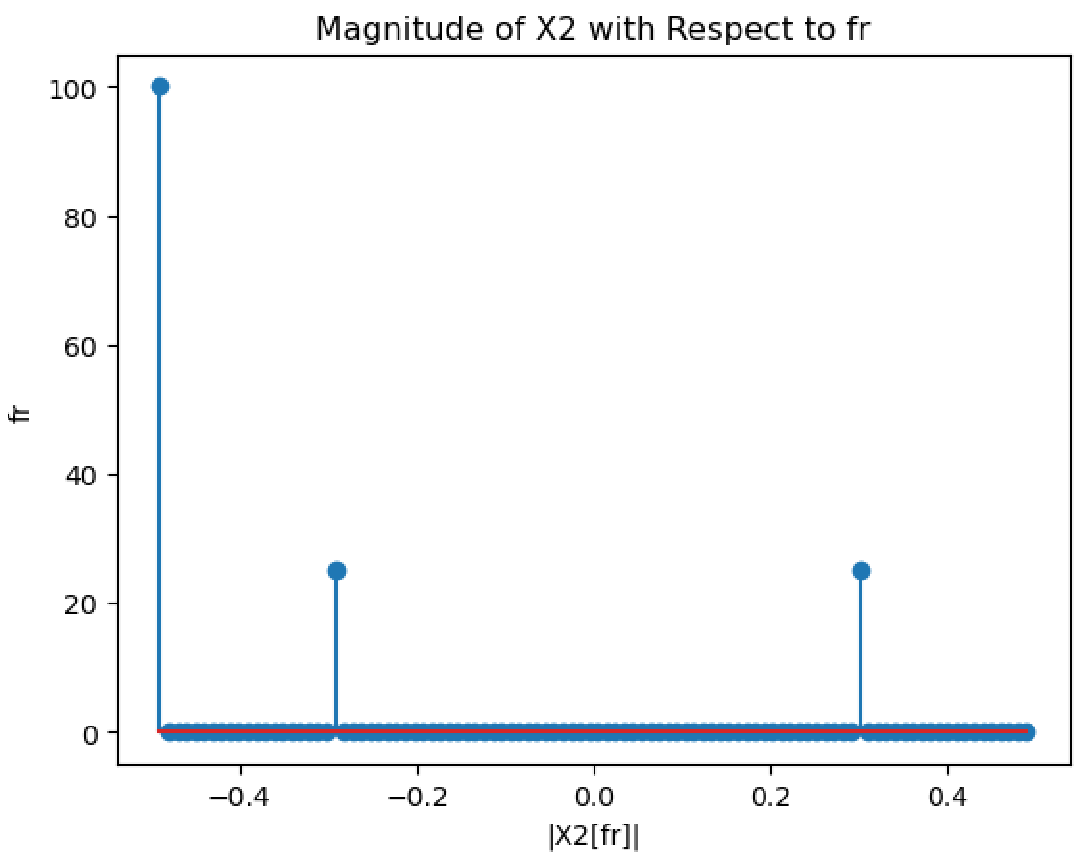




Yes, we see improvement here, as the graph seems perfectly symmetrical now, and it is centered exactly at the origin, instead of slightly to one side.

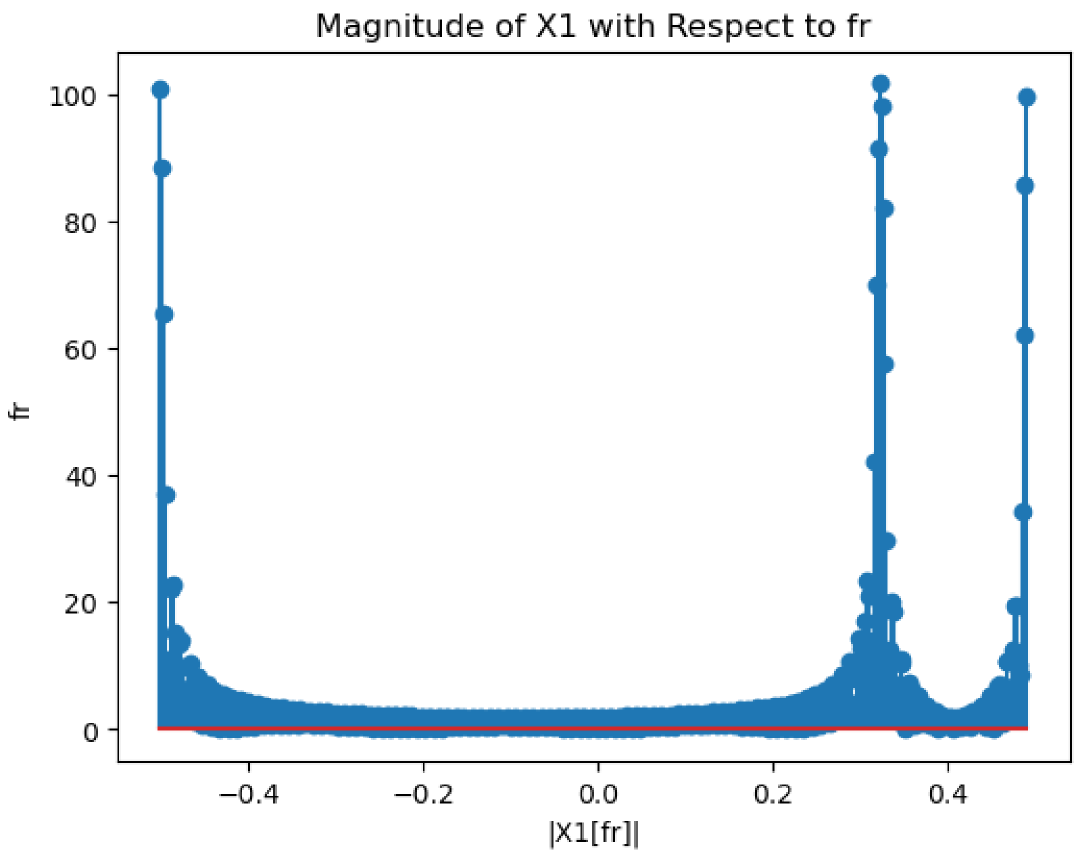
[10]:
```
#A3

import numpy as np
import matplotlib.pyplot as plt

n = np.arange(0, 100)
N = 100
fr = np.linspace(-0.49, 0.49, N)
x1 = np.exp(1j*2*np.pi*((10*5)/100)*n) + np.exp(1j*2*np.pi*(33/100)*n)
x2 = np.cos(2*np.pi*((10*5)/100)*n) + 0.5*np.cos(2*np.pi*((10*3)/100)*n)

X3 = np.fft.fftshift(np.fft.fft(x1))
X4 = np.fft.fftshift(np.fft.fft(x2))

plt.stem(fr, abs(X3))
plt.title("Magnitude of X1 with Respect to fr")
plt.xlabel('|X1[fr]|')
plt.ylabel('fr')
plt.show()
plt.stem(fr, abs(X4))
plt.title("Magnitude of X2 with Respect to fr")
plt.xlabel('|X2[fr]|')
plt.ylabel('fr')
plt.show()
```

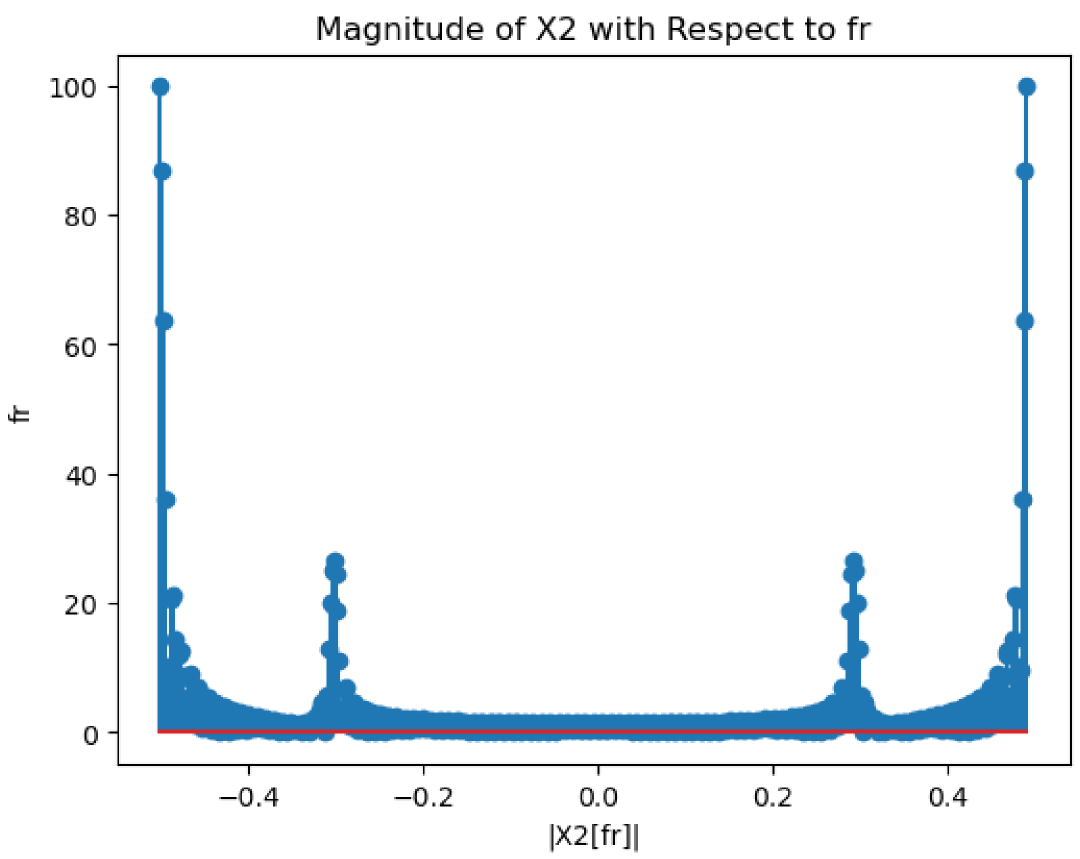




Since X2 is even, it is symmetrical.

[11]:
```
#A4

import numpy as np
import matplotlib.pyplot as plt

n = np.arange(0, 101)
N = 501
fr = np.linspace(-0.5, 0.49, N)
x1 = np.exp(1j*2*np.pi*((10*5)/100)*n) + np.exp(1j*2*np.pi*(33/100)*n)
x3 = np.concatenate((np.zeros(200), x1, np.zeros(200)))
x2 = np.cos(2*np.pi*((10*5)/100)*n) + 0.5*np.cos(2*np.pi*((10*3)/100)*n)
x4 = np.concatenate((np.zeros(200), x2, np.zeros(200)))

X3 = np.fft.fftshift(np.fft.fft(x3))
X4 = np.fft.fftshift(np.fft.fft(x4))

plt.stem(fr, abs(X3))
plt.title("Magnitude of X1 with Respect to fr")
plt.xlabel('|X1[fr]|')
plt.ylabel('fr')
plt.show()

plt.stem(fr, abs(X4))
plt.title("Magnitude of X2 with Respect to fr")
plt.xlabel('|X2[fr]|')
plt.ylabel('fr')
plt.show()
```





Yes, there is improvement as it is more ccurate now, with more points plotted closer together. This gives us a full signal here, appears perfectly symmetrical and centered at 0 (for X2 at least).

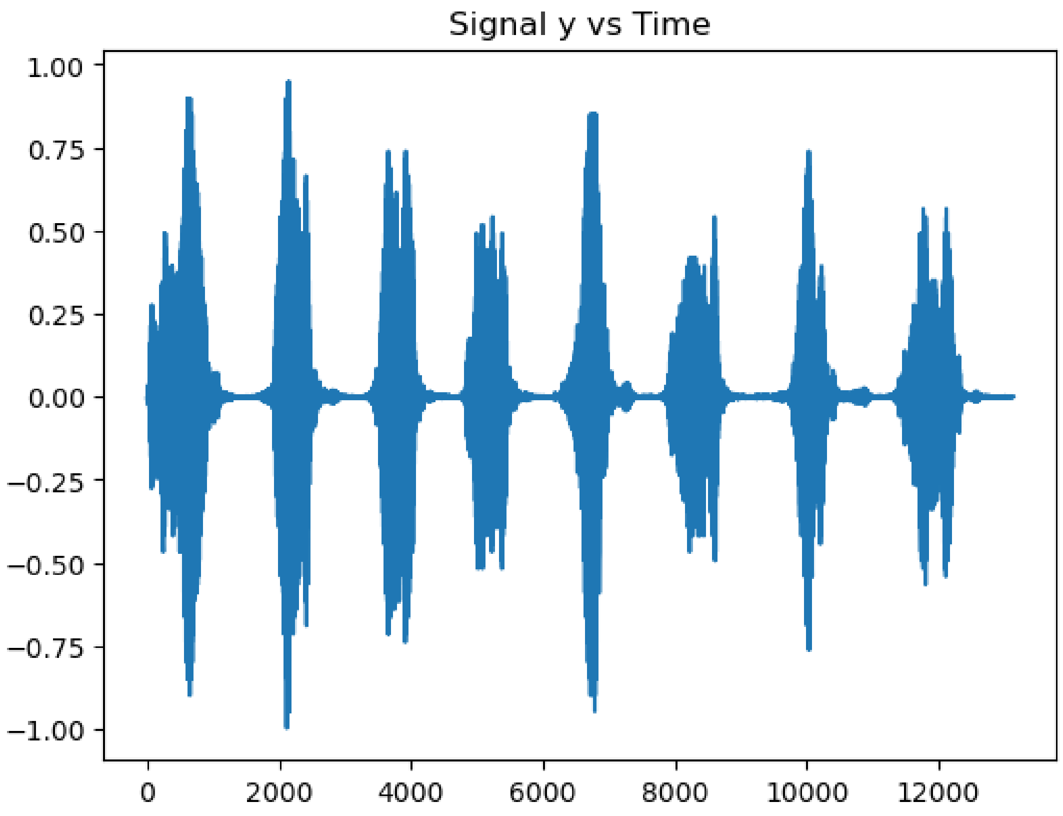
2 —-Part B—-

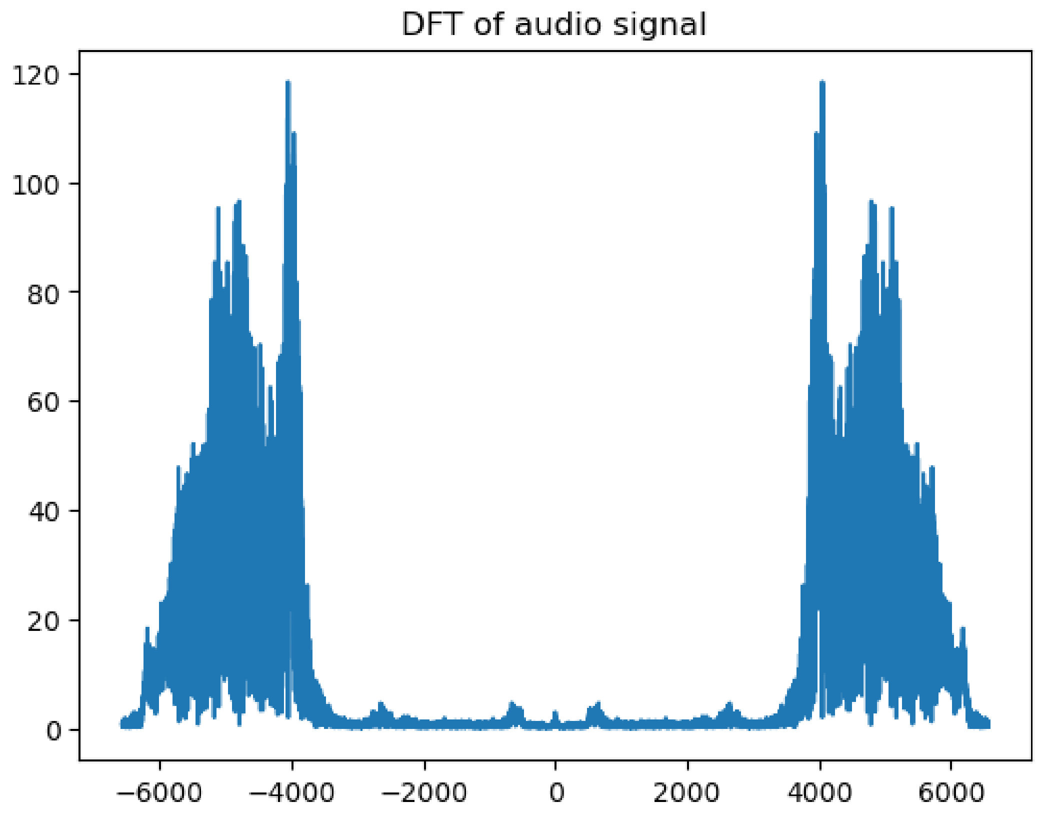
[8]:
```
#Should have all parts here below
import numpy as np
import scipy.io.wavfile as wav
import matplotlib.pyplot as plt
import soundfile
import sounddevice as sd

filename = 'chirp.wav'
y, fs = soundfile.read(filename)

No = len(y)
To = len(y) / fs
Ti = 1 / fs

#Number of Samples
print("No = ",No)
```

```
#Duration of Signal
print("To = ",To)

#Sampling Interval
print("Ti = ",Ti)

t = np.arange(No)
plt.plot(t, y)
plt.title('Signal y vs Time')
plt.show()


Y = np.fft.fftshift(np.fft.fft(y))
fr = np.arange(-No/2, No/2)

plt.plot(fr, np.abs(Y))
plt.title('DFT of audio signal')
plt.show()


rate = 2
y1 = y[::rate]
No2 = len(y1)
To2 = 2 * len(y1) / fs
Ti2 = 2 / fs

#Number of Samples
print("No = ",No2)

#Duration of Signal
print("To = ",To2)

#Sampling Interval
print("Ti = ",Ti2)


plt.plot(y1)
plt.title('Signal y vs Time')
plt.show()


Y1 = np.fft.fftshift(np.fft.fft(y1))
fr = np.arange(-No2/2, No2/2)

plt.plot(fr, np.abs(Y1))
plt.title('DFT of audio signal')
```

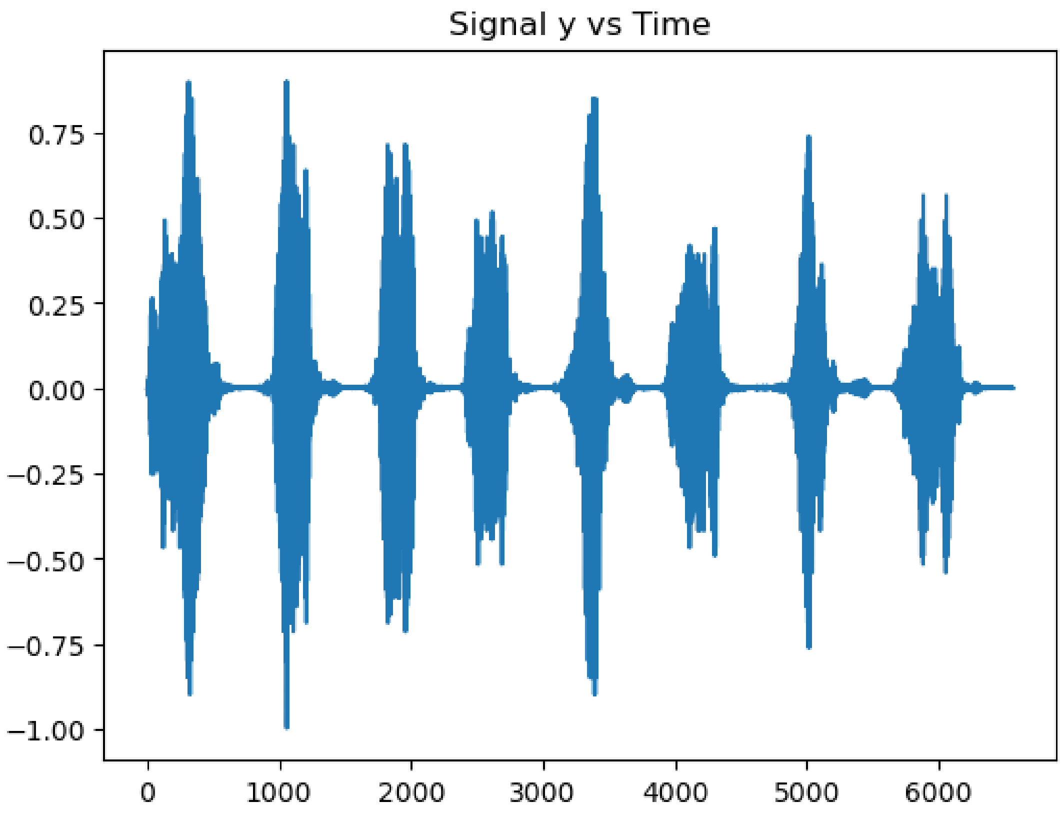
```python
plt.show()


from IPython.display import Audio
Audio(data=y, rate=fs)
Audio(data=y1, rate=fs)


rate = 5
y2 = y[::rate]
No3 = len(y2)
To3 = 2 * len(y2) / fs
Ti3 = 2 / fs

#Number of Samples
print("No = ",No3)

#Duration of Signal
print("To = ",To3)

#Sampling Interval
print("Ti = ",Ti3)

plt.plot(y2)
plt.title('Signal y vs Time')
plt.show()

Y2 = np.fft.fftshift(np.fft.fft(y2))
fr = np.arange(-No3/2, No3/2)

plt.plot(fr, np.abs(Y2))
plt.title('DFT of audio signal')
plt.show()

Audio(data=y, rate=fs)
```

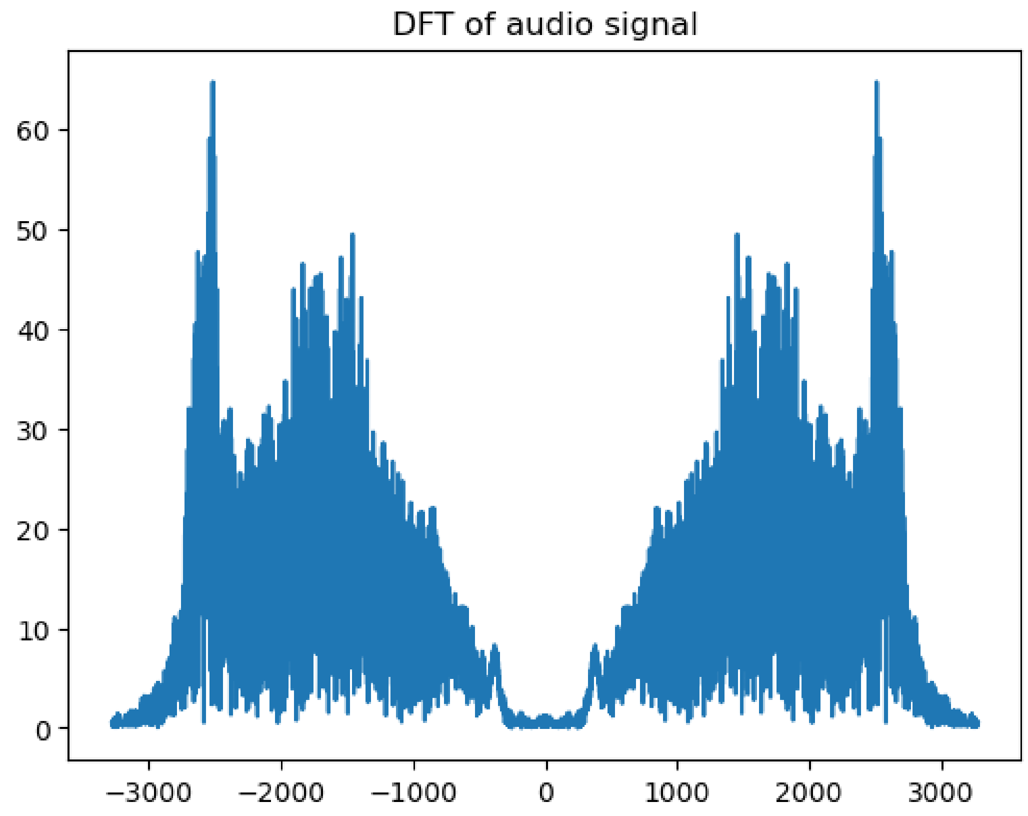
```
No =  13129
To =  1.6026611328125
Ti =  0.0001220703125
```


Signal y vs Time
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2000
4000
6000
8000
10000
12000

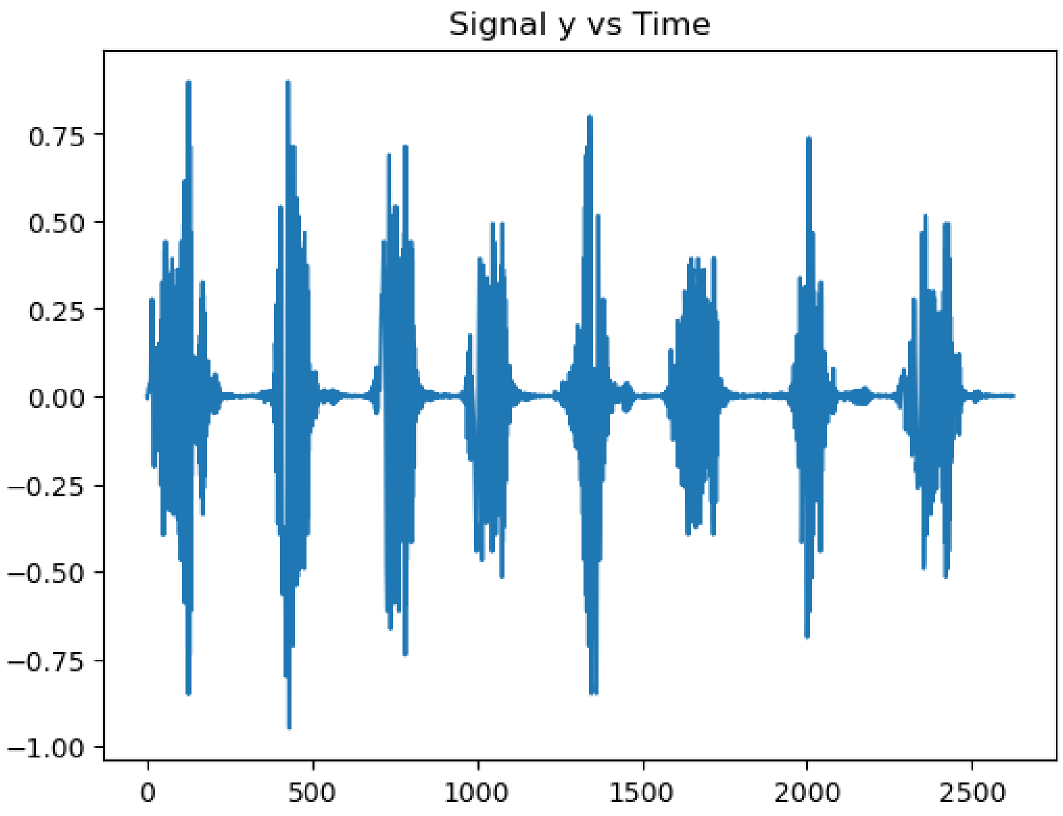


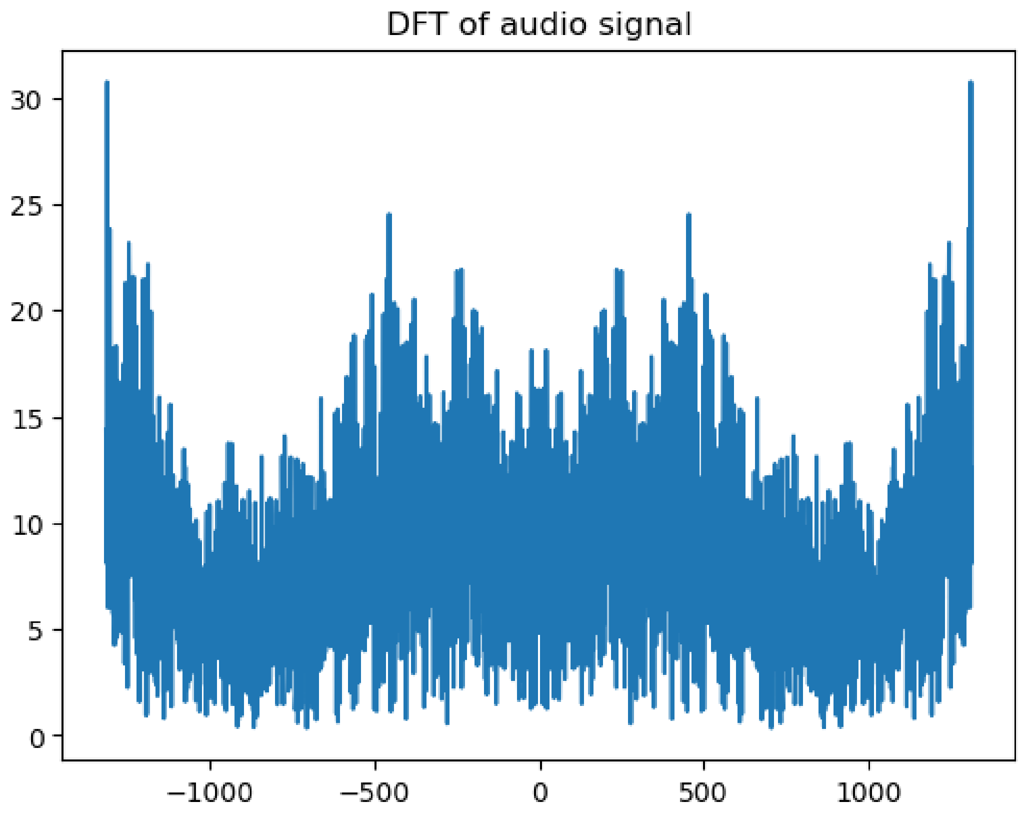
```
No =  6565
To =  1.602783203125
Ti =  0.000244140625
```


Signal y vs Time
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
1000
2000
3000
4000
5000
6000



```
No =  2626
To =  0.64111328125
Ti =  0.000244140625
```


Signal y vs Time
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
500
1000
1500
2000
2500



```
[8]: <IPython.lib.display.Audio object>
```

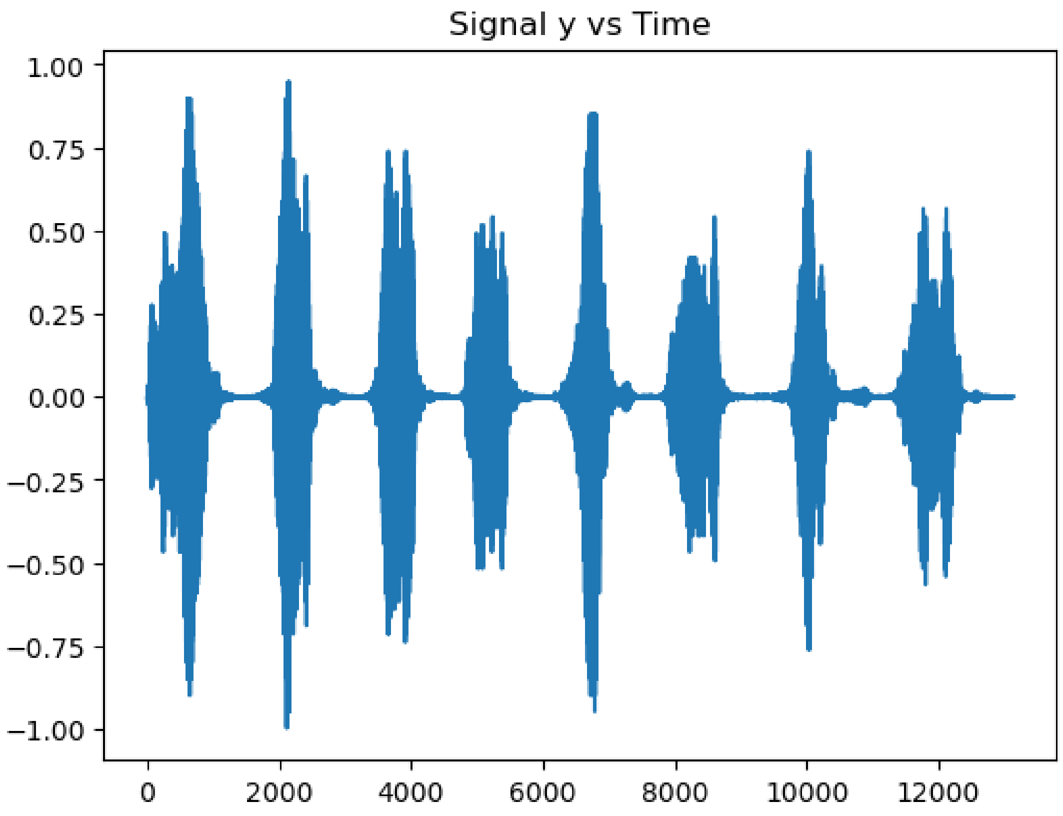
3 Part C———-

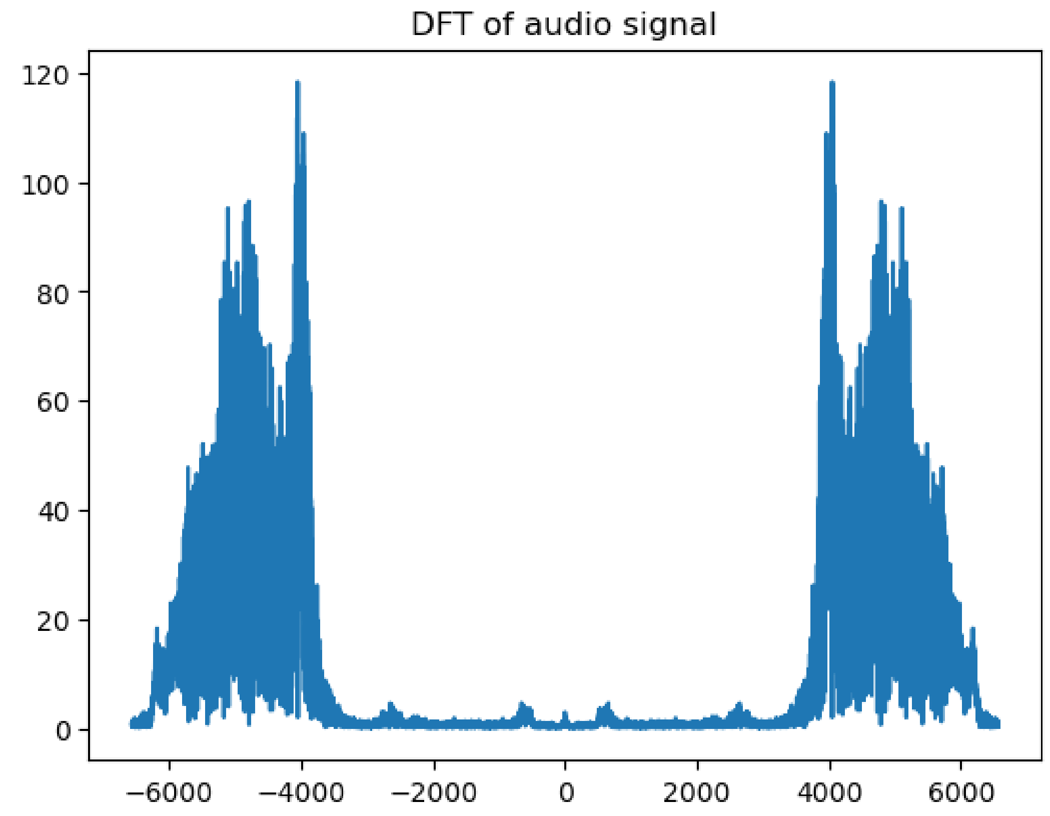
```
[9]: import numpy as np
     import scipy.io.wavfile as wav
     import matplotlib.pyplot as plt
     import soundfile
     import sounddevice as sd
     from IPython.display import Audio

     filename = 'chirp.wav'
     y, fs = soundfile.read(filename)

     No = len(y)
     To = len(y) / fs
     Ti = 1 / fs

     #Number of Samples
     print("No = ",No)
```

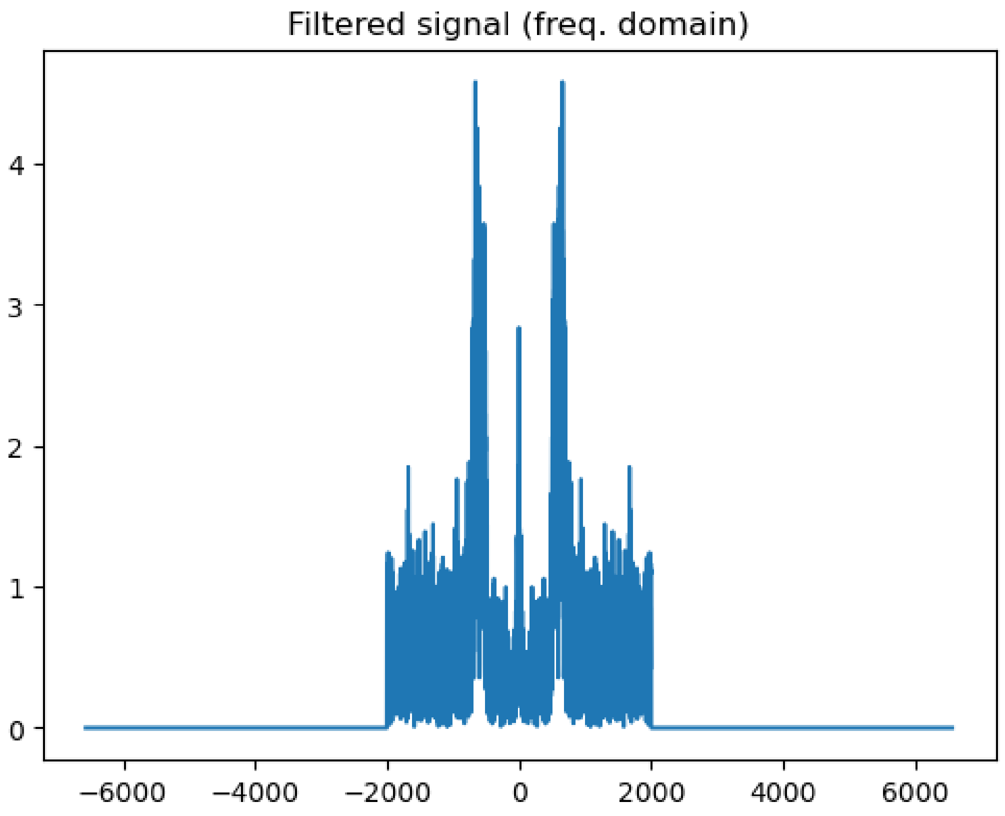
```
#Duration of Signal
print("To = ",To)

#Sampling Interval
print("Ti = ",Ti)

t = np.arange(No)
plt.plot(t, y)
plt.title('Signal y vs Time')
plt.show()

Y = np.fft.fftshift(np.fft.fft(y))
fr = np.arange(-No/2, No/2)

plt.plot(fr, np.abs(Y))
plt.title('DFT of audio signal')
plt.show()

# Rectangular filter that only passes frequencies less than 2000
rect_filter = np.ones_like(Y)
rect_filter[np.abs(fr) > 2000] = 0

# Apply the filter
Y_filtered = Y * rect_filter

# Plot the filtered signal in the frequency
plt.plot(fr, np.abs(Y_filtered))
plt.title('Filtered signal (freq. domain)')
plt.show()

# Get the filtered signal in the time
y_filtered = np.real(np.fft.ifft(np.fft.ifftshift(Y_filtered)))

# Plot the filtered signal in the time
plt.plot(t, y_filtered)
plt.title('Filtered signal (time domain)')
plt.show()

# Play the filtered signal
Audio(data=y_filtered, rate=fs)

# Design a filter that cuts the frequencies between 16 and 256 Hz
bass_filter = np.ones_like(Y)
bass_filter[(np.abs(fr) > 16) & (np.abs(fr) < 256)] = 0

# Apply filter
```

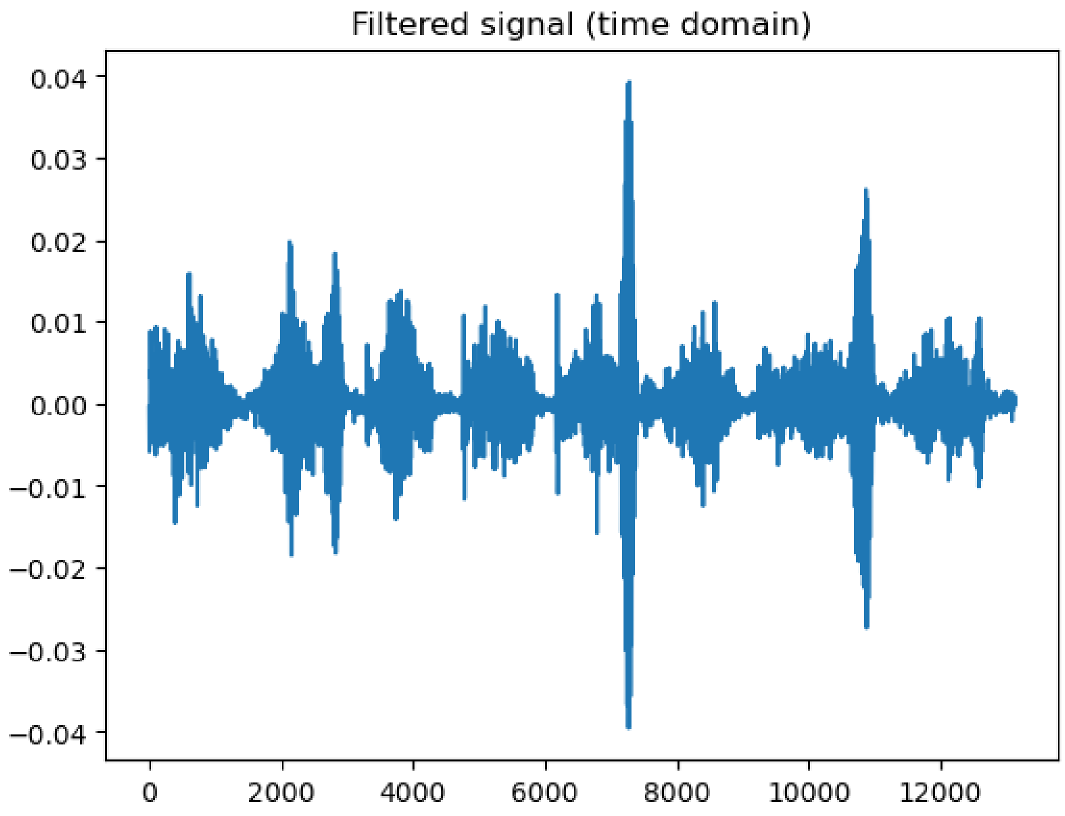
```
Y_bass_filtered = Y * bass_filter

# Plot the filtered signal in the frequency
plt.plot(fr, np.abs(Y_bass_filtered))
plt.title('Filtered signal (freq. domain)')
plt.show()

# Get the filtered signal in the time
y_bass_filtered = np.real(np.fft.ifft(np.fft.ifftshift(Y_bass_filtered)))

# Plot the filtered signal in the time
plt.plot(t, y_bass_filtered)
plt.title('Filtered signal (time domain)')
plt.show()

# Play the filtered signal
Audio(data=y_bass_filtered, rate=fs)

# Design a filter that amplifies frequencies between 2048 and 16384 Hz by 25%
treble_filter = np.ones_like(Y)
treble_filter[(np.abs(fr) > 2048) & (np.abs(fr) < 16384)] *= 1.25

# Apply the filter
Y_treble_filtered = Y * treble_filter

# Plot the filtered signal in the frequency
plt.plot(fr, np.abs(Y_treble_filtered))
plt.title('Filtered signal (freq. domain)')
plt.show()

# Get the filtered signal in the time
y_treble_filtered = np.real(np.fft.ifft(np.fft.ifftshift(Y_treble_filtered)))

# Plot the filtered signal in the time
plt.plot(t, y_treble_filtered)
plt.title('Filtered signal (time domain)')
plt.show()

# Play the filtered signal
Audio(data=y_treble_filtered, rate=fs)
```

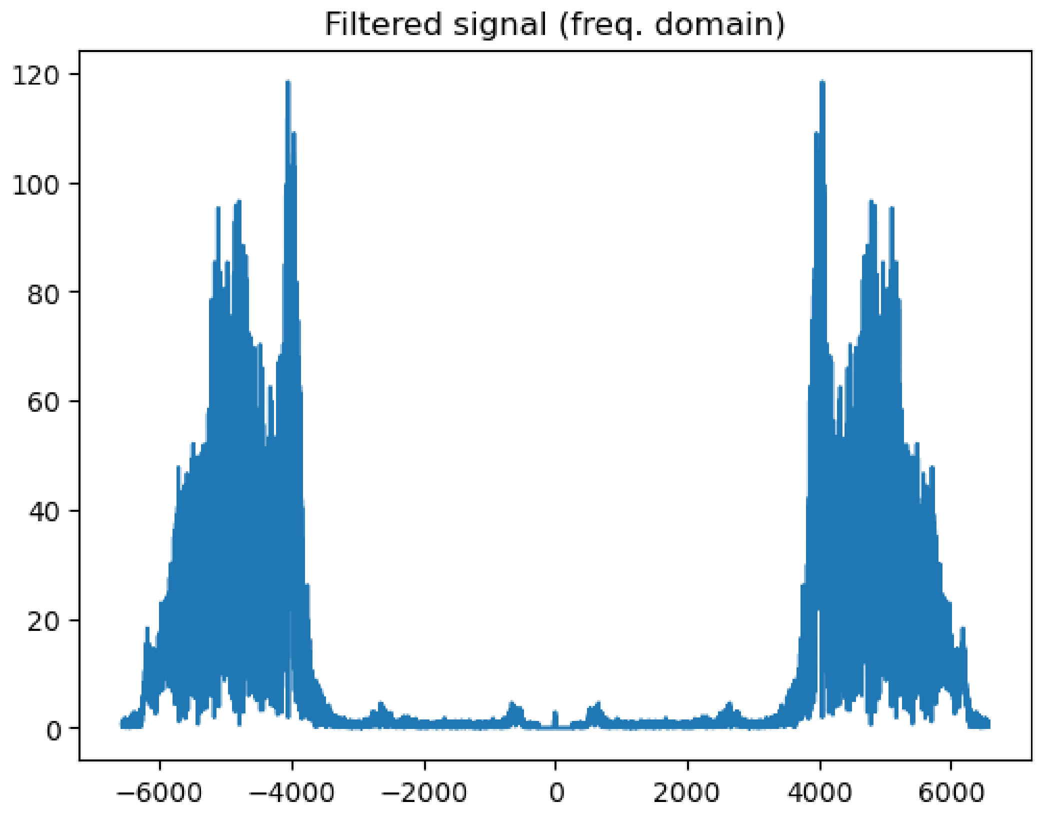
```
No =  13129
To =  1.6026611328125
Ti =  0.0001220703125
```

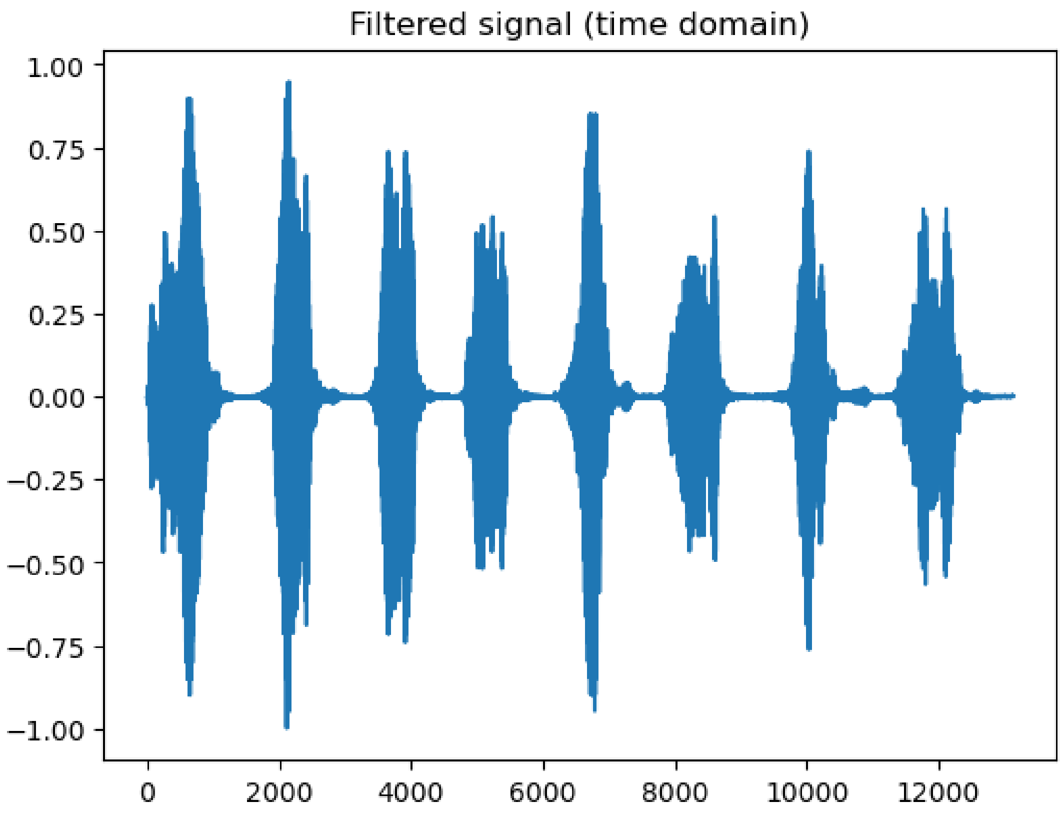

Signal y vs Time
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2000
4000
6000
8000
10000
12000

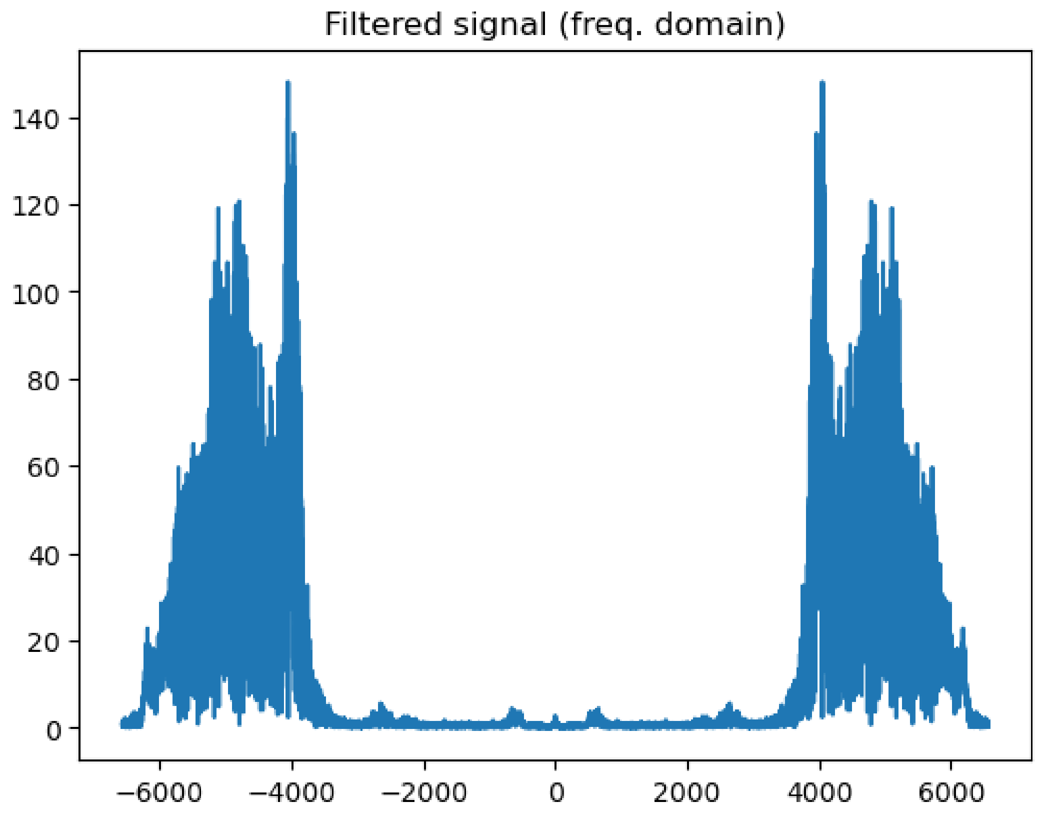

DFT of audio signal
120
100
80
60
40
20
0
−6000
−4000
−2000
0
2000
4000
6000

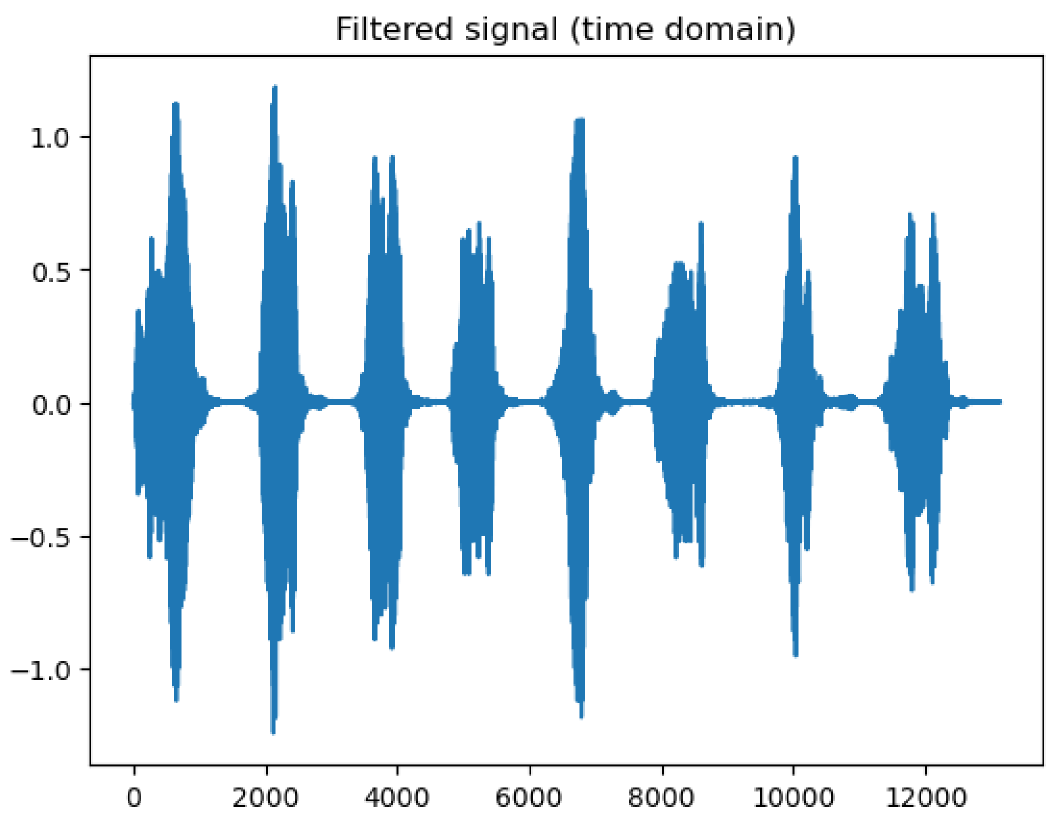

Filtered signal (freq. domain)
0
1
2
3
4
−6000
−4000
−2000
0
2000
4000
6000


Filtered signal (time domain)
0.04
0.03
0.02
0.01
0.00
−0.01
−0.02
−0.03
−0.04
0
2000
4000
6000
8000
10000
12000


Filtered signal (freq. domain)
120
100
80
60
40
20
0
−6000
−4000
−2000
0
2000
4000
6000


Filtered signal (time domain)
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
2000
4000
6000
8000
10000
12000


Filtered signal (freq. domain)
140
120
100
80
60
40
20
0
−6000
−4000
−2000
0
2000
4000
6000



```
[9]: <IPython.lib.display.Audio object>
```

```
[10]: #Part 5)

      #We used the DFT property where multiplication in one domain is convolution in␣
       ↪the other and vice-versa. We multiplied
      #the filter with the output signal to get the filtered spectrum of the signal.
```

```
[ ]: filepath = "C:\\Users\\zelen\\ELE632_lab5_DaniloZelenovic_501032542_Section08.
      ↪ipynb"
     !jupyter nbconvert --to pdf $filepath
```