Course Code: CSE 4632

Course Name: Digital Signal Processing Lab

Lab No: 4

Name: Md Farhan Ishmam

ID: 180041120

Lab Group: P

Task-1

Explanation: We import the necessary libraries. Then the audio is loaded. We check the sample rate and pick first 50,000 samples. Using the numpy `rfft()` function we perform fast fourier analysis. Afterwards, we take the maximum magnitude and convert it to the corresponding frequency, and hence, get the main frequency of the tuning fork.

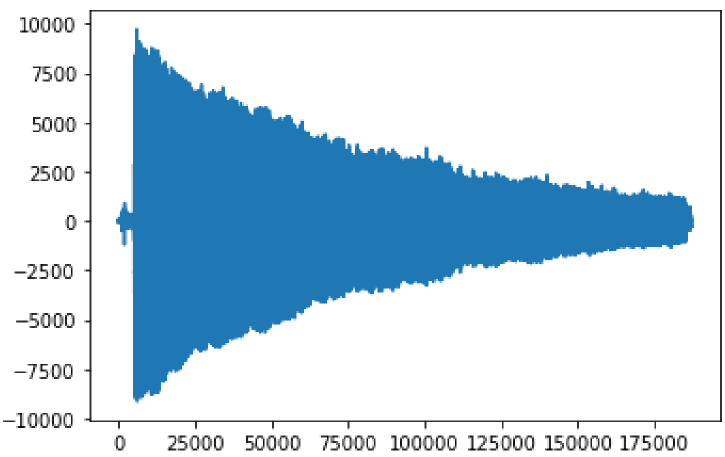
```
#Importing the necessary libraries
from IPython.display import Audio
from scipy.io import wavfile
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
#Loading the audio file
samplerate, data = wavfile.read('/content/Lab 4.wav')
```

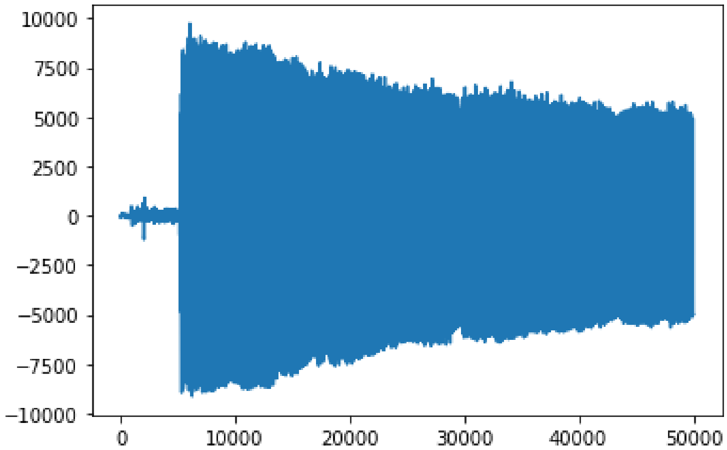
```
print(samplerate)
```

```
22050
```

```
plt.plot(data)
plt.show()
```



```
#Taking first 50,000 samples
data = data[0:50000]
plt.plot(data)
plt.show()
```



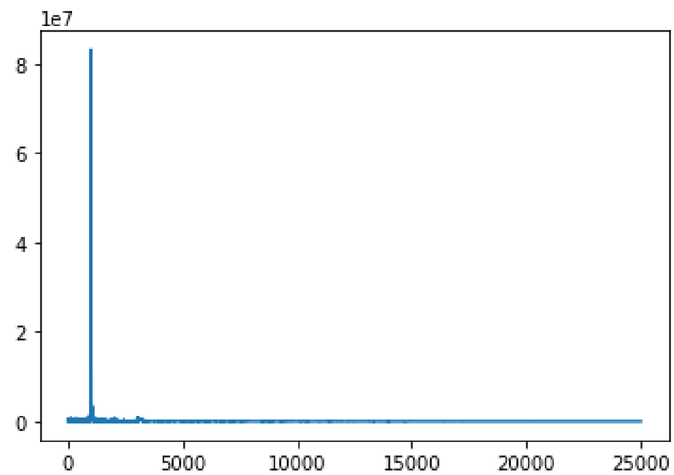
```
#Performing rfft and getting the real and imaginary parts
ReX = np.fft.rfft(data).real
ImX = -np.fft.rfft(data).imag
```

```
print(ReX, ImX)
```

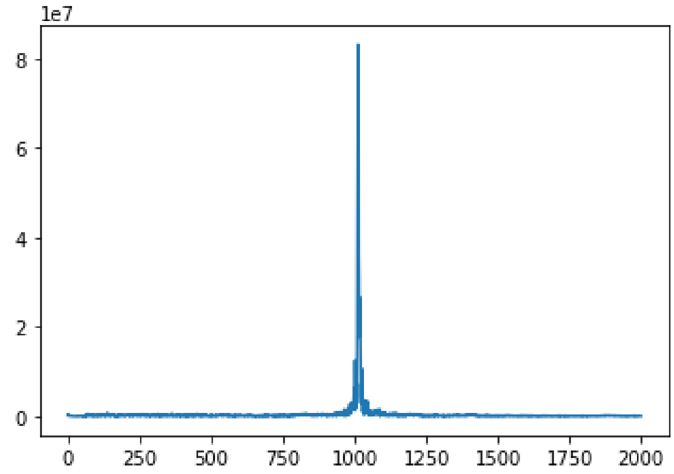
```
[502334.          95148.84190888 148077.14507715 ...   2406.67978325
   2418.34416522   2422.        ] [-0.00000000e+00 -9.57288411e+04  2.02787171e+05 ... -1.6
 -7.36935371e+00 -0.00000000e+00]
```

```
#Obtaining the magnitudes from real and imaginary parts
MgX = np.sqrt(np.square(ReX) + np.square(ImX))
```

```
plt.plot(MgX)
plt.show()
```



```
plt.plot(MgX[0:2000])
plt.show()
```




```
#Taking the maximum magnitude
MgMax = MgX.argmax()
print(MgMax)
```

```
1012
```

```
#Converting the maximum magnitude to frequency
#First sample is zero and last sample is 0.5 of sampling rate
freq = (MgMax/25000)*0.5*samplerate
print(freq)
```

```
446.29200000000003
```

Spectral Analysis Results

The frequency of the tuning fork is 446 Hz

Task-2

Explanation: DFT requires us to implement the given equations. We implement this using two loops to iterate over the dataset elements and perform cos and sin functions using `np.cos()` and `np.sin()`.

$$ReX[k] = \sum_{i=0}^{N-1} x[i] \cos(2\pi k i / N)$$

$$ImX[k] = -\sum_{i=0}^{N-1} x[i] \sin(2\pi k i / N)$$

```
def myDFT(x):
  n = len(x)
  ReX = np.zeros(int(n/2+1))
```

```
  ImX = np.zeros(int(n/2+1))
  for k in range(len(ReX)):
    for i in range(n):
      ReX[k] = ReX[k] + x[i]*np.cos((2*np.pi*k*i)/n)
      ImX[k] = ImX[k] - x[i]*np.sin((2*np.pi*k*i)/n)
  return ReX, ImX
```

Explanation: Inverse DFT requires synthesis of signal from the real and imaginary parts. Looping over the real and imaginary parts, we first scale the values using the following equations.

$$Re\bar{X}[k] = \frac{ReX[k]}{N/2}$$

$$Im\bar{X}[k] = -\frac{ImX[k]}{N/2}$$

except for two special cases:

$$Re\bar{X}[0] = \frac{ReX[0]}{N}$$

$$Re\bar{X}[N/2] = \frac{ReX[N/2]}{N}$$

Afterwards, we perform synthesis by summming over the real and imaginary values multiplied to their cosine and sine counterparts respectively. The following equation is used for synthesis and finding the value of x which will be returned as the output.

$$x[i] = \sum_{k=0}^{N/2} Re\bar{X}[k]\cos(2\pi ki/N) + \sum_{k=0}^{N/2} Im\bar{X}[k]\sin(2\pi ki/N)$$

```
def myIDFT(ReX, ImX):

  n = len(ReX) + len(ImX) - 2
  x = np.zeros(n)

  for k in range(len(ReX)):
    ReX[k] = ReX[k]/(n/2)
    ImX[k] = ImX[k]/(n/2)
  ReX[0] = ReX[0]/2
  ReX[len(ReX)-1] = ReX[len(ReX)-1]/2

  for k in range(len(ReX)):
    for i in range(n):
```

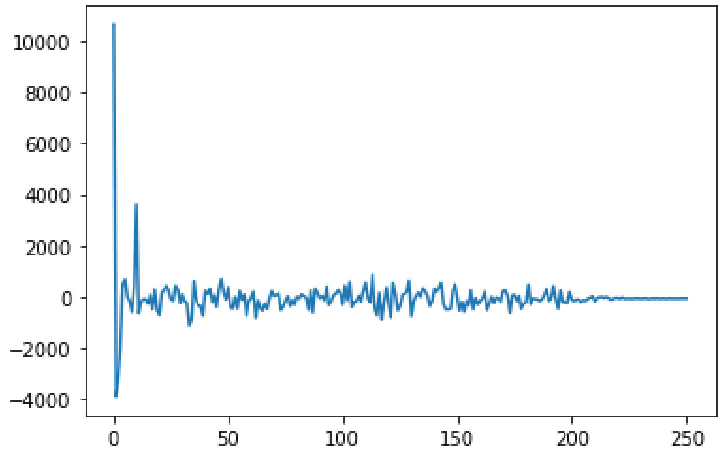
```
            x[i] = x[i] + ReX[k]*np.cos((2*np.pi*k*i)/n)
            x[i] = x[i] - ImX[k]*np.sin((2*np.pi*k*i)/n)
    return x
```

Now, we compare the DFT and IDFT functions with the numpy implementations.

```
newData = data[0:500]
myReX, myImX = myDFT(newData)
ReX = np.fft.rfft(newData).real
ImX = -np.fft.rfft(newData).imag
```

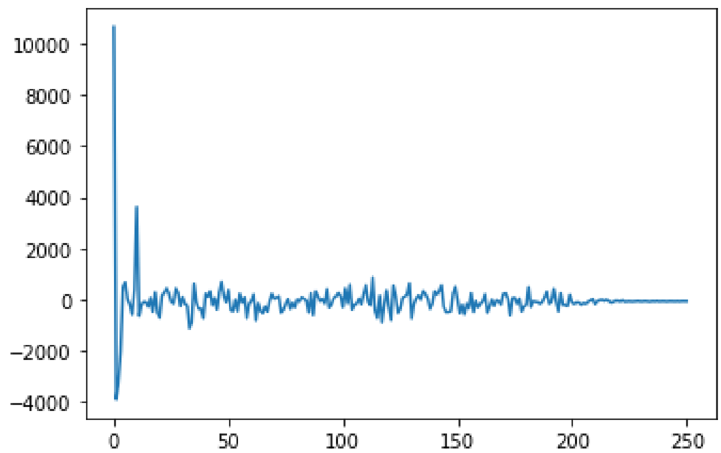
```
#Graph of the real part of our DFT function
plt.plot(myReX)
```

```
[<matplotlib.lines.Line2D at 0x7f04dfdb2110>]
```



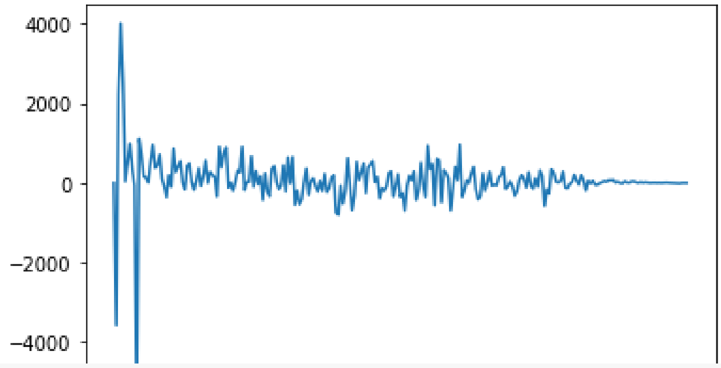

```
#Graph of the real part of numpy's FFT function
plt.plot(ReX)
```

```
[<matplotlib.lines.Line2D at 0x7f04dff046d0>]
```



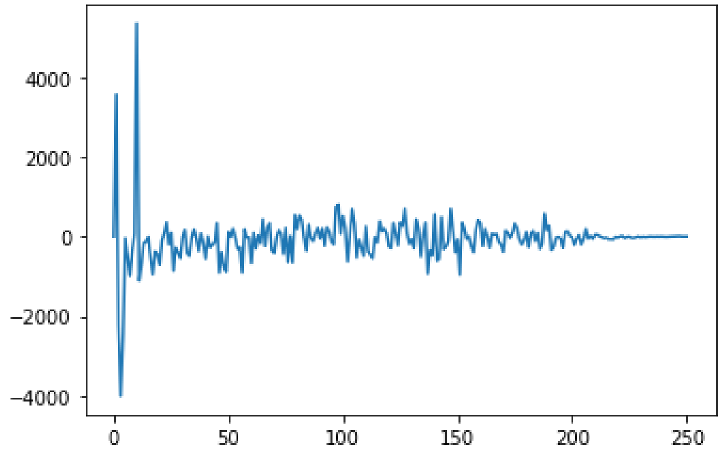

```
#Graph of the imaginary part of our DFT function
plt.plot(myImX)
```

```
[<matplotlib.lines.Line2D at 0x7f04dfcf5350>]
```



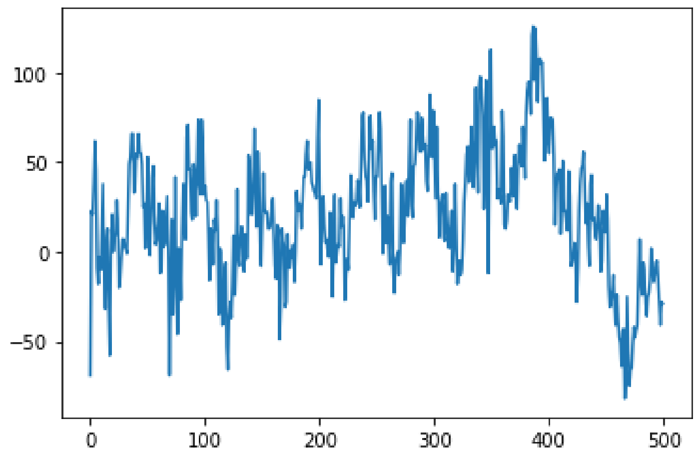

```
#Graph of the imaginary part of numpy's FFT function
plt.plot(ImX)
```

```
[<matplotlib.lines.Line2D at 0x7f04dfc5d150>]
```



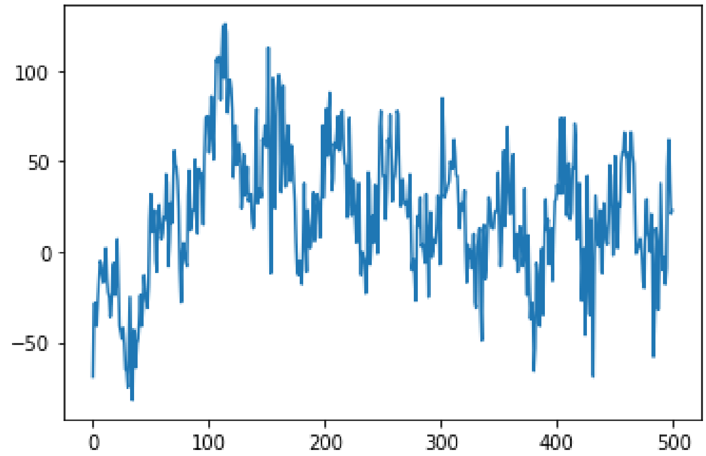

```
#Graph of our implementation of inverse DFT function
xIDFT = myIDFT(ReX, ImX)
plt.plot(xIDFT)
```

```
[<matplotlib.lines.Line2D at 0x7f04dfc484d0>]
```




```
#Graph of the original signal
plt.plot(newData)
```

```
[<matplotlib.lines.Line2D at 0x7f04dfba59d0>]
```



Our implementation of DFT and IDFT has results similar to numpy's FFT implementation.