# PROCESSING OF BENGALI SPEECH SIGNAL FOR EXTRACTION AND STUDY OF PARAMETERS TARGETED FOR VOWEL RECOGNITION

*In team:*
*Annasha Ghosh & Shubham*
*Interns:University of Calcutta*

# A Short Introduction To Automatic Speech Recognition:

Information processing machines have become ubiquitous. However, the current modes of human machine communication are geared more towards living with the limitations of computer input/output devices rather than the convenience of humans. Speech is the primary mode of communication among human beings. On the other hand, prevalent means of input to computers is through a keyboard or a mouse. It would be nice if computers could listen to human speech and carry out their commands. Automatic Speech Recognition (ASR) is the process of deriving the transcription (word sequence) of an utterance, given the speech waveform. Speech understanding goes one step further, and gleans the meaning of the utterance in order to carry out the speaker's command.

In this work,speech samples of utterances of 'a' , 'e' , 'o' ,'u' have been collected from seven(7) subjects.The main target of the work is identification of features that can distinguish utterance of one vowel from the other irrespective of the speaker.

Our work has proceeded in the following phases:

1. Understanding of WAV/RIFF signal and its acquisition.
2. Preprocessing the signal
    - Low Pass Filter
3. Finding periodicity of a Speech waveform
    - Correlation
    - Normalization
    - Maximization
    - Finding peak values
    - Obtaining a threshold value
    - Dynamic window and template formation
4. Zero Crossing Count(ZCC) over one period for different vowels
5. Table for ZCC for different subjects for each vowel
6. Comparison of vowel waveform for different subjects

7. Vowel Synthesis

# 1.***Understanding of WAV/RIFF signal and its acquisition***.

## ***Wav description:***

The WAVE file format is a subset of Microsoft's RIFF specification for the storage of multimedia files. A RIFF file starts out with a file header followed by a sequence of data chunks. A WAVE file is often just a RIFF file with a single "WAVE" chunk which consists of two sub-chunks -- a "fmt " chunk specifying the data format and a "data" chunk containing the actual sample data. Call this form the "Canonical form". Who knows how it really works. An almost complete description which seems totally useless unless you want to spend a week looking over it can be found at MSDN (mostly describes the non-PCM, or registered proprietary data formats).

## ***Wav file format:***

The canonical WAVE format starts with the RIFF header:

| | | | |
|---|---|---|---|
| 0 | 4 | **ChunkID** | Contains the letters "RIFF" in ASCII form (0x52494646 big-endian form). |
| 4 | 4 | **ChunkSize** | 36 + SubChunk2Size, or more precisely: 4 + (8 + SubChunk1Size) + (8 + SubChunk2Size) This is the size of the rest of the chunk following this number. This is the size of the entire file in bytes minus 8 bytes for the two fields not included in this count: ChunkID and ChunkSize. |
| 8 | 4 | **Format** | Contains the letters "WAVE" (0x57415645 big-endian form). |

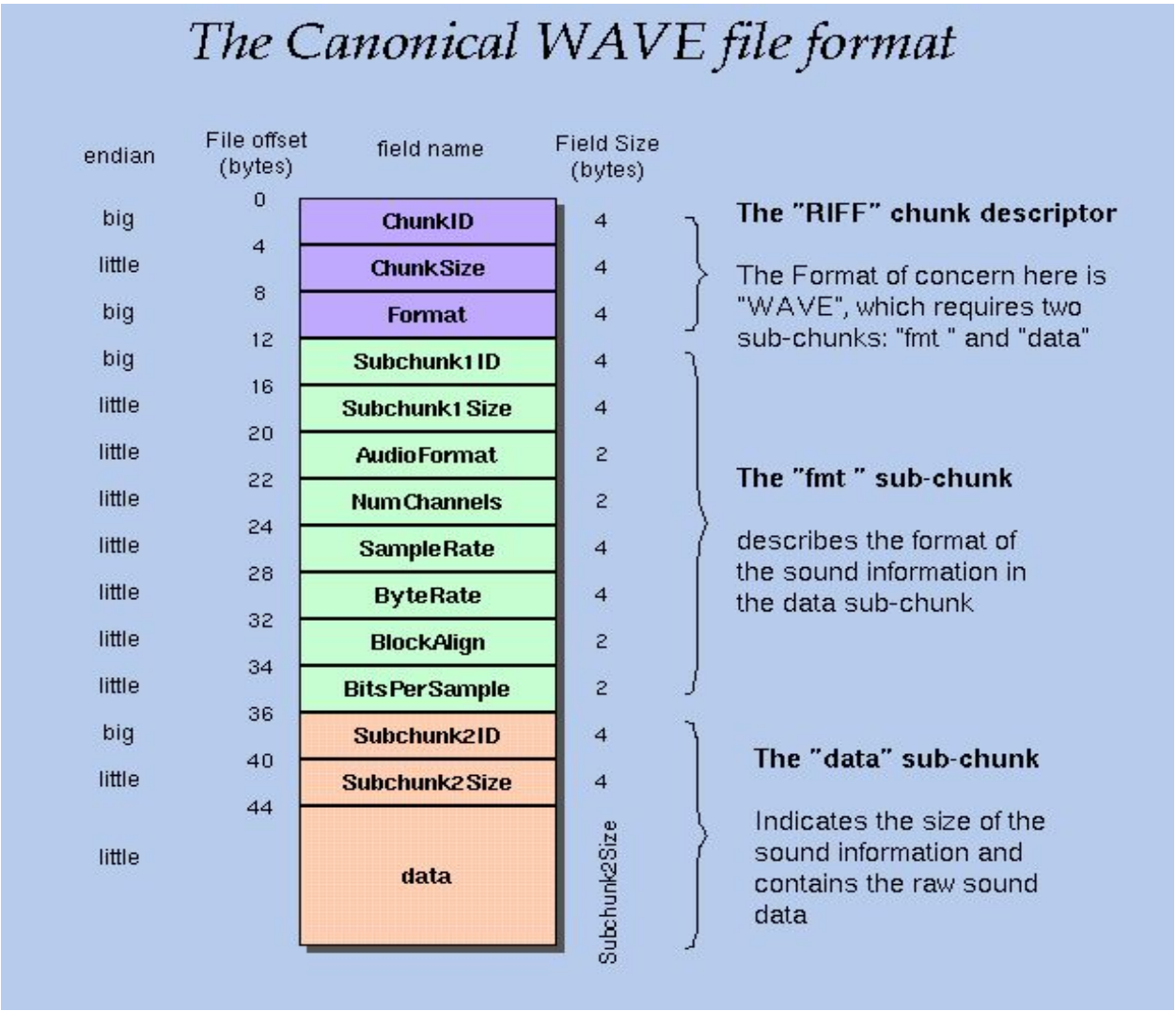
The "WAVE" format consists of two **subchunks**: "fmt " and "data":
The "fmt " subchunk describes the sound data's format:

| | | | |
|---|---|---|---|
| 12 | 4 | **Subchunk1ID** | Contains the letters "fmt " (0x666d7420 big-endian form). |

| | | | |
|---|---|---|---|
| 16 | 4 | **Subchunk1Size** | 16 for PCM. This is the size of the rest of the Subchunk which follows this number. |
| 20 | 2 | **AudioFormat** | PCM = 1 (i.e. Linear quantization) Values other than 1 indicate some form of compression. |
| 22 | 2 | **NumChannels** | Mono = 1, Stereo = 2, etc. |
| 24 | 4 | **SampleRate** | 8000, 44100, etc. |
| 28 | 4 | **ByteRate** == | SampleRate * NumChannels * BitsPerSample/8 |
| 32 | 2 | **BlockAlign** == | NumChannels * BitsPerSample/8 The number of bytes for one sample including all channels. |
| 34 | 2 | **BitsPerSample** | 8 bits = 8, 16 bits = 16, etc. |
| | 2 | **ExtraParamSize** | if PCM, then doesn't exist |
| | X | **ExtraParams** | space for extra parameters |

The "data" subchunk contains the size of the data and the actual sound:

| | | | |
|---|---|---|---|
| 36 | 4 | **Subchunk2ID** | Contains the letters "data" (0x64617461 big-endian form). |
| 40 | 4 | **Subchunk2Size** == | NumSamples * NumChannels * BitsPerSample/8 This is the number of bytes in the data. You can also think of this as the size of the read of the subchunk following this Number. |
| 44 | * | **Data** | The actual sound data. |




Src: http://soundfile.sapp.org/doc/WaveFormat/

❖ ***Understanding of Wav file format using Java:***

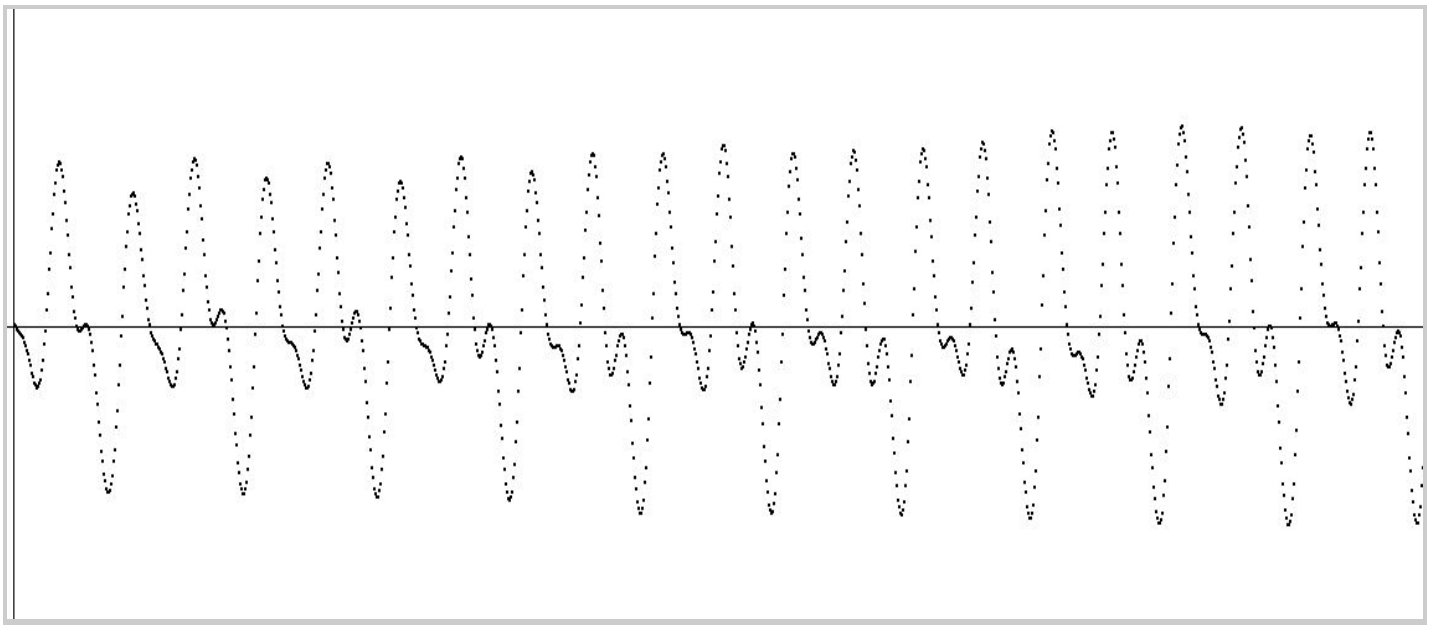
Using 'InputStream' wav file was stored in Byte array.Information was extracted from its header and was visualized.relevant information can be extracted by masking respective bytes.Information required for plotting the data are :

1. Sample rate
2. Audio format
3. Number of channels
4. Data

```
import java.awt.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.*;

public class myFile extends Frame {
   static byte[] audioBytes;

   myFile(byte[] audioBytes){
        this.audioBytes = audioBytes;

        setVisible(true);
        setLayout(new FlowLayout());
        setLocation(200,200);
        setSize(1000,700);
        addWindowListener(new WindowAdapter() {
                        public void windowClosing(WindowEvent e) {
                        System.exit(0);
            }
        });
}
   myFile(){
        setVisible(true);
        setLayout(new FlowLayout());
        setSize(1000,700);
        addWindowListener(new WindowAdapter() {
                        public void windowClosing(WindowEvent e) {
                      System.exit(0);
            }
        });
 }
 public void paint(Graphics g){
       g.drawLine(40,400,2000,400);
        g.drawLine(50,0,50,1000);
        int i=45; int j=0;
        for ( i=205,j=0; i<audioBytes.length;i+=10,j++) {
            g.drawOval(50+j, 400+audioBytes[i], 1, 1);
        }
   }
   public static void main (String[] args){
        try {
            ByteArrayOutputStream out = new ByteArrayOutputStream();
            BufferedInputStream in = new BufferedInputStream(new FileInputStream("filename"));
              int read;
            byte[] buff = new byte[1024];
            while ((read = in.read(buff)) > 0) {
                out.write(buff, 0, read);
            }
            out.flush();
            audioBytes = out.toByteArray();
```

```
            System.out.println(audioBytes.length);
            for (int i=45 ; i<audioBytes.length;i++){
                System.out.println(audioBytes[i]);
            }
        }
        catch (Exception e){

        }
        myFile my = new myFile(audioBytes);

    }
}
```

## :Plotted data of wav file using Java:



❖ ***Extraction of data from wav files and its operations using Python:***

Using Python libraries namely *numpy,matplotlib,scipy and math* we have tried to explore various wav files and observe their characteristic traits.The wav files were mainly audio files containing bengali and hindi vowels recorded by various people.

```
import numpy as np
from scipy.signal import butter, lfilter, freqz
from matplotlib import pyplot as plt
from scipy.io import wavfile
from scipy.fftpack import fft,fftfreq
from matplotlib import pyplot as plt
import math
import scipy.signal as signal
```

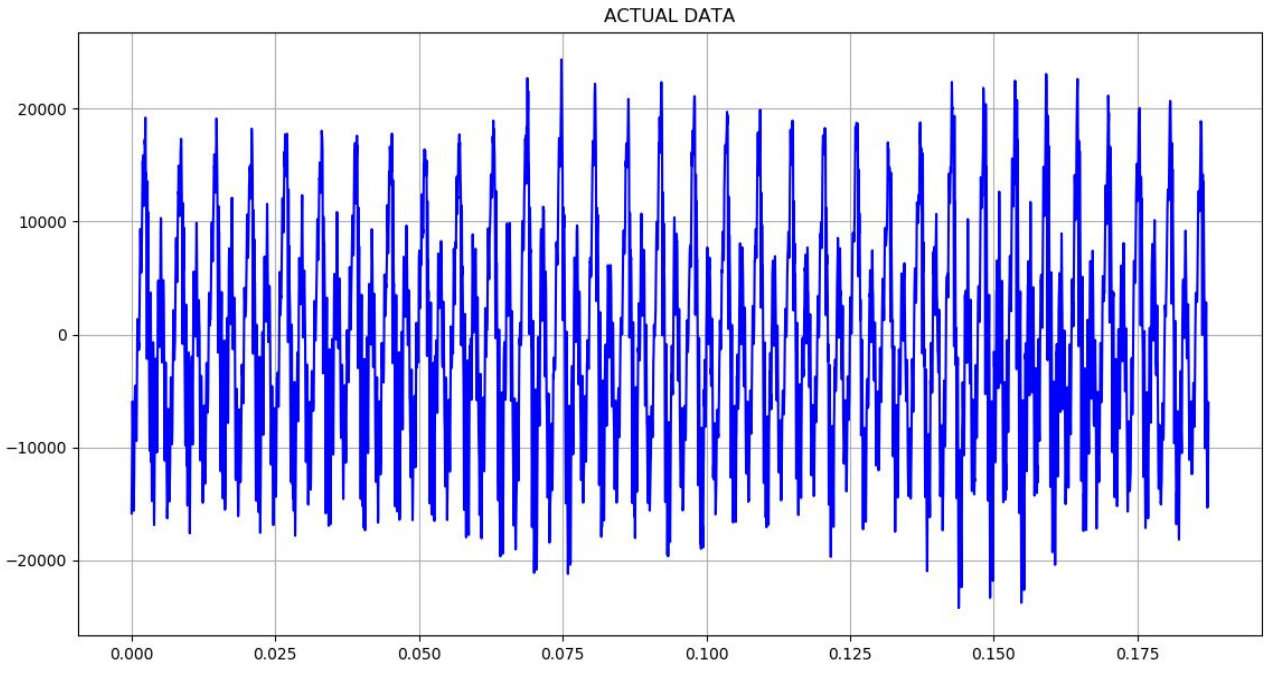
A step-wise analysis and conclusions are detailed below.

❖ *Reading of wav files:*

The wav files are read.Due to the canonical form of the wav file,its sample rate & actual sample data can be extracted. Thus the sample rate and actual sample data are extracted stored in python variable and list respectively.The total number of samples of a given wav file can also be noted.

The actual sample data is then plotted to get a clear visualization of the signal.

```
begin:
        samplerate , data = wavefile.read('file_name.wav')
        data = array storing the actual sample data
        samples = data.shape[0]              //obtaining the total number of samples
        Time_period = samples/samplerate
        X_axis = np.linspace(0 , Time_period , samples)
        Y_axis = data
        plt.plot(X_axis , Y_axis , 'b-')
        plt.show()
end
```



# 2. ***Preprocessing the signal***

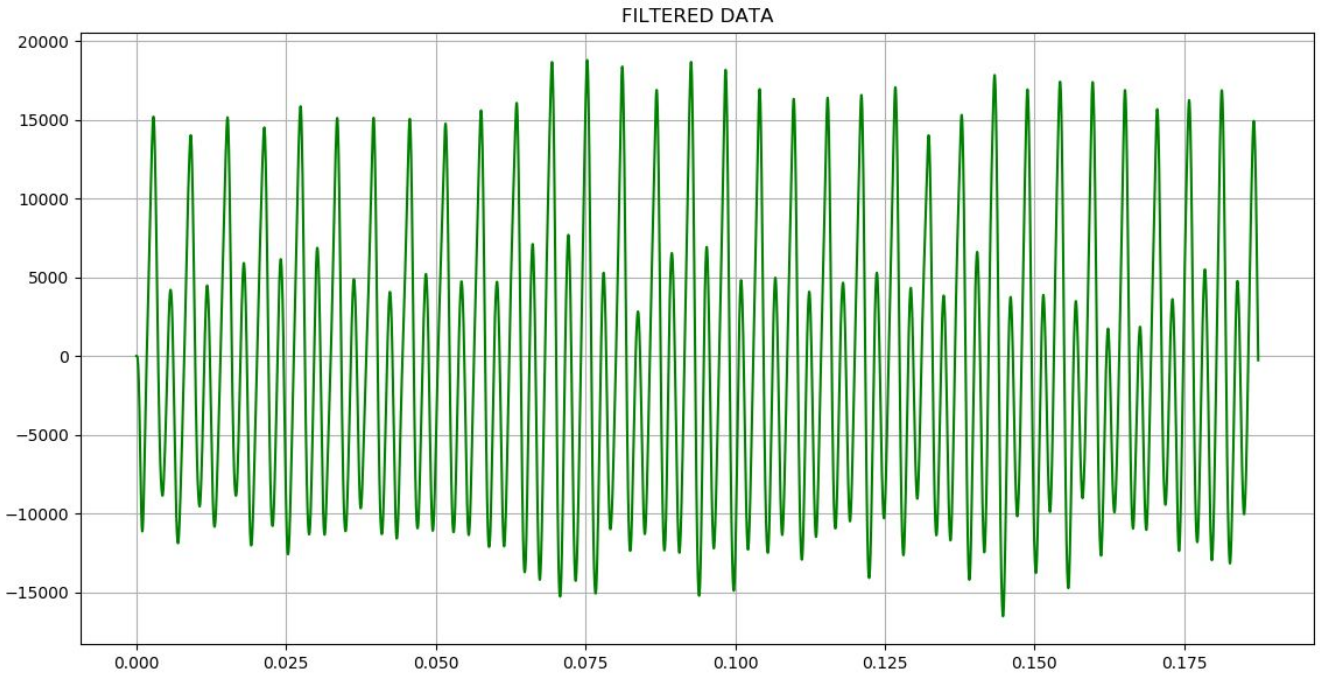
- *Low Pass Filtering of the extracted data:*

  The actual sample data that is extracted is now filtered using Low Pass Filtering to get a noise devoid signal.

  Two functions, *butter_lowpass() and butter_lowpass_filter()*, are explicitly used to filter the actual sample data so that further operations can be carried out over a noiseless signal.

  Again the filtered data is plotted for visualization of the noiseless signal.

```
begin:
        butter_lowpass (cut-off, samplerate, order=5)
                nyq = 0.5 * samplerate
                normal_cutoff = cut-off / nyq
                //obtaining the filtering coefficients
                b, a = butter (order, normal_cutoff, btype='low', analog =False)
                return b, a
         butter_lowpass_filter (data, cut-off, samplerate, order=5)
                 b, a = butter_lowpass (cut-off, fs, order=order)
                 filtered_data = lfilter (b, a, data )          //obtaining the filtered data
                return filtered_data
        filtered_data = butter_lowpass_filter (data, cut-off, samplerate, order)
        X_axis = np.linspace( 0 , Time_period , samples)
        Y_axis = filtered_data
        plt.plot(X_axis , Y_axis , 'g-')
        plt.show()
end
```




## 3.***Finding periodicity of a Speech waveform***

**a.)** *Correlation:*

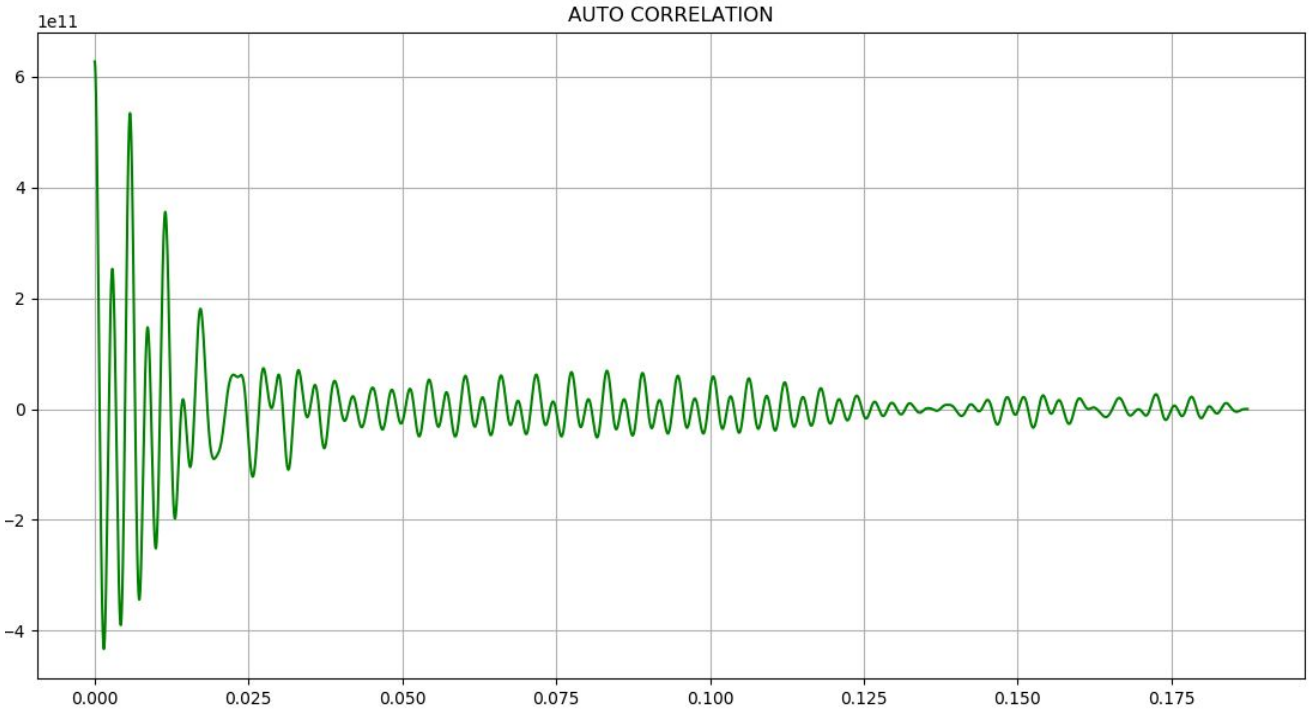
From the entire signal,a window is formed of a fixed length based on mentioned time units.Then a template is formed from the window which is again of a fixed length based on another mentioned time units.The window is correlated with the template.

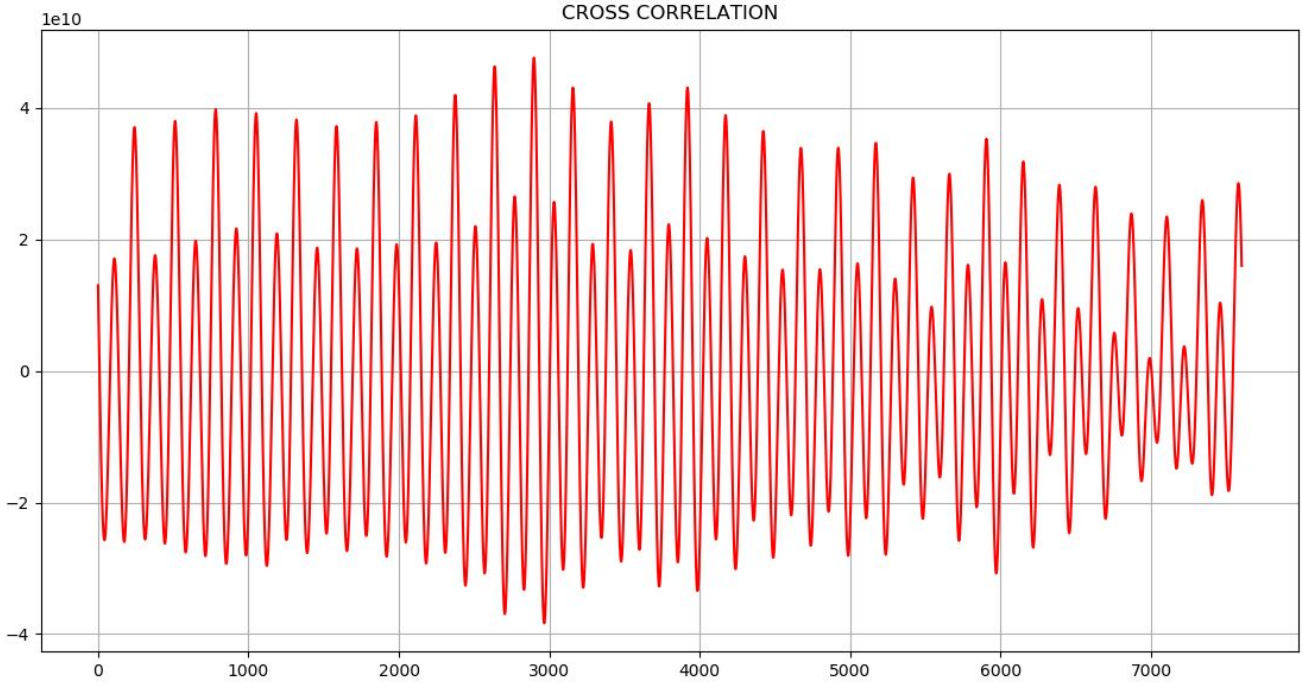
If the template size and the window size are equal then we perform *auto-correlation* otherwise *cross_correlation*.

Whenever two wave signal is correlated we achieve dominant peaks when the signal of the window exactly matches with that of the template.Thus Correlation helps in understanding the periodicity of any given signal.

```
begin:
        n=samples
        wave_correlation (window, template):
                len_window = length of window
                len_template = length of template
                r = list ();          // creating a list to store the correlated data
                //AUTO-CORRELATION
                if (len_window==len_template)
                        for i from 0 to len_window
                        n=0
                        for j from i to len_window
                                        n = n + (window[j]*window[j-i])
                r.append (int(n))
                else:
                //CROSS-CORRELATION
                        l=len_template
                        w=0
                        while True:
                n=0
                for i, j from (w to l), (0 to len_template)
                                n=n + window[i] * template[j]
                r.append(int(n))
                w=w+1
```

```
                        l=l+1
                        if(l>len_window)
                                        break
        return r
                plt.plot(wave_correlation(window, template))
                plt.show()
        end
```


AUTO CORRELATION
1e11
6
4
2
0
−2
−4
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175

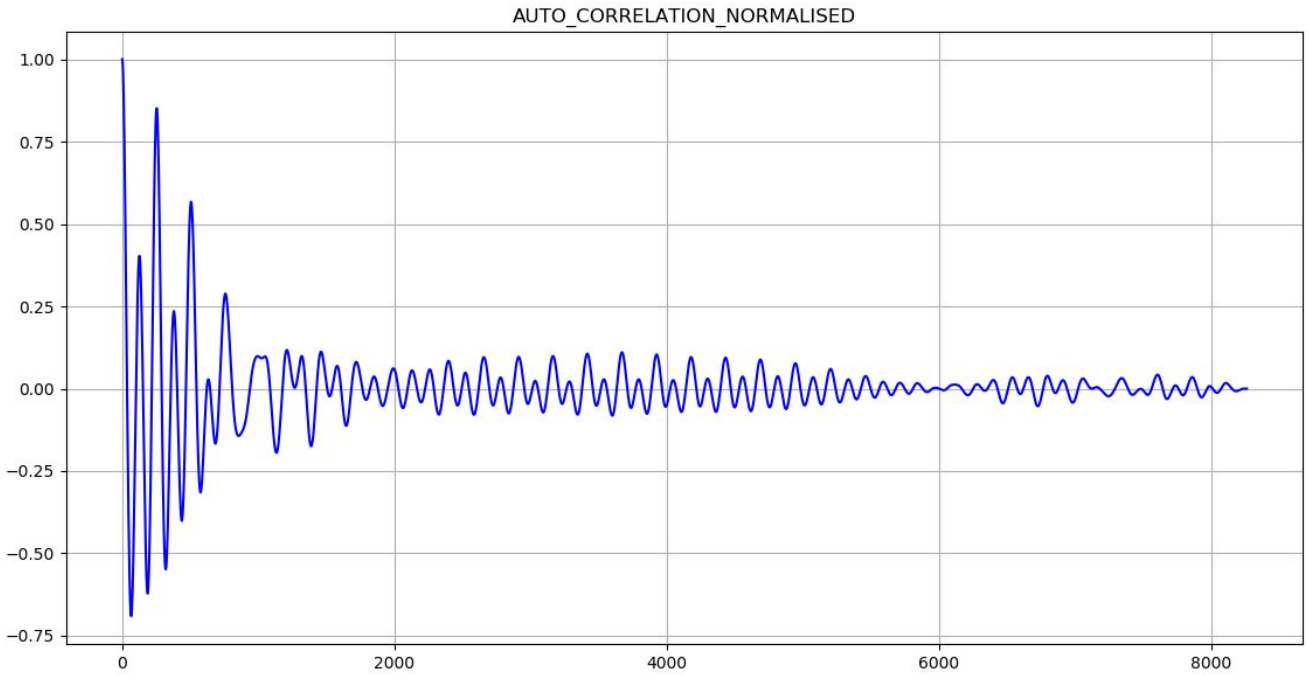

CROSS CORRELATION
1e10
4
2
0
−2
−4
0
1000
2000
3000
4000
5000
6000
7000

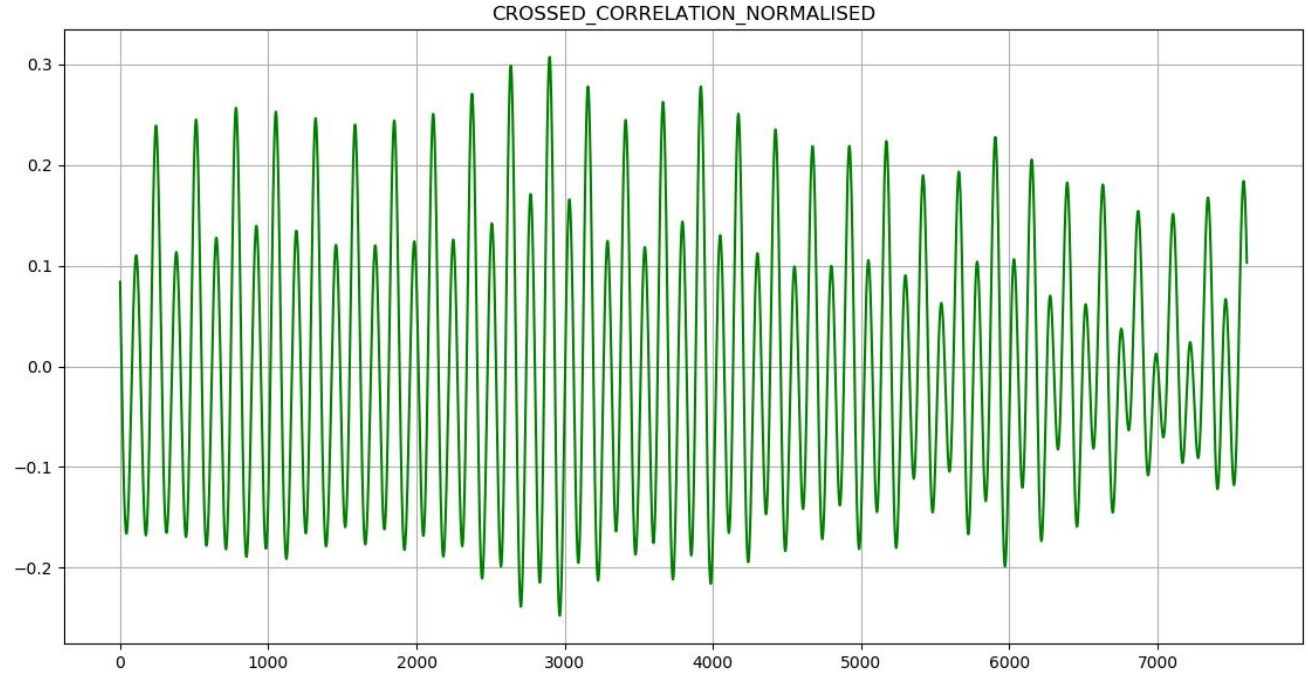
**b.)** *Normalisation:*

Correlation does not signify whether the window and the template formed are of the same signal or not.Thus *normalisation* becomes an integral part of the experiment to ensure that the signals that are correlated are similar.

Normalisation value is close to 1 if the two signals are same otherwise it is close to 0.

```
begin:
        wave_normalization(window , template)
                len_window = length of window
                len_template = length of template
                a=0
                for i from 0 to len_window
                        a = a + math.pow(window[i],2)
                b=0
                for j from 0 to len_template
                        b = b + math.pow(template[j],2)
                Normalising_factor = np.sqrt(a*b)
                r=wave_correlation(window, template)
                r1=list();
                for i from 0 to (length of r )
                        n = r[i] / Normalising_factor
                        r1.append(n)
                return r1
        plt.plot(wave_normalization(window, template))
        plt.show()
end
```

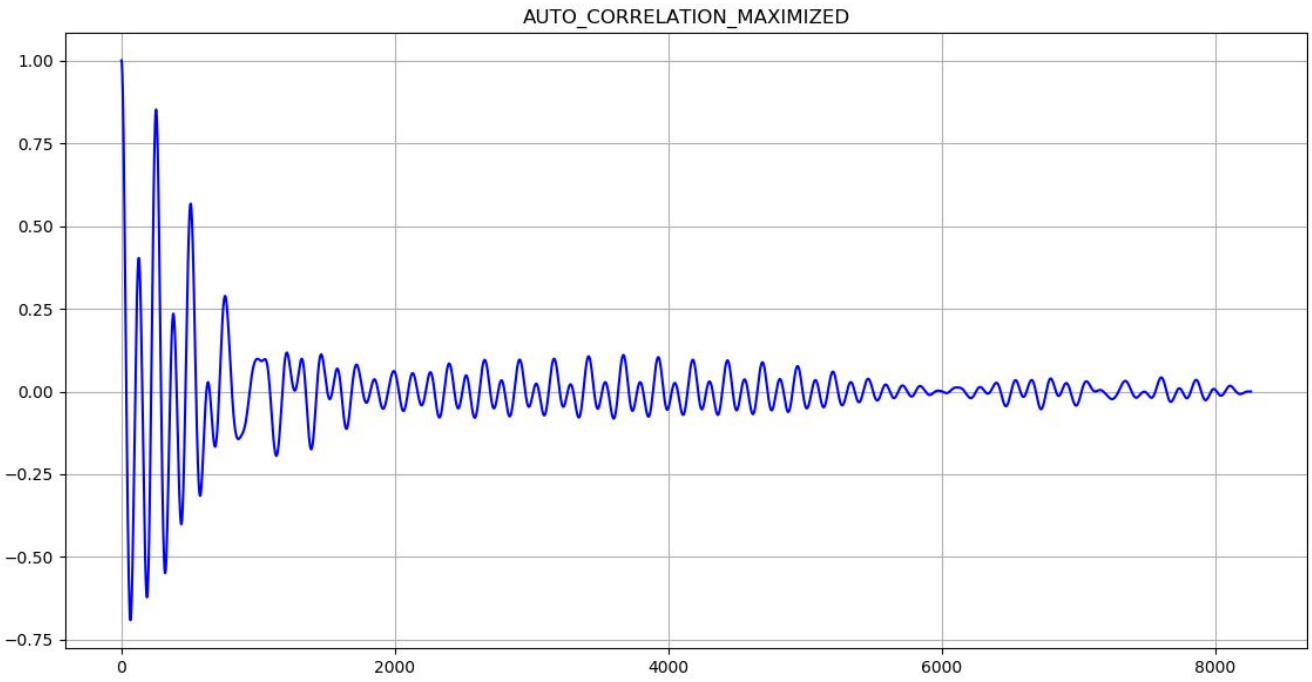

AUTO_CORRELATION_NORMALISED
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
0
2000
4000
6000
8000

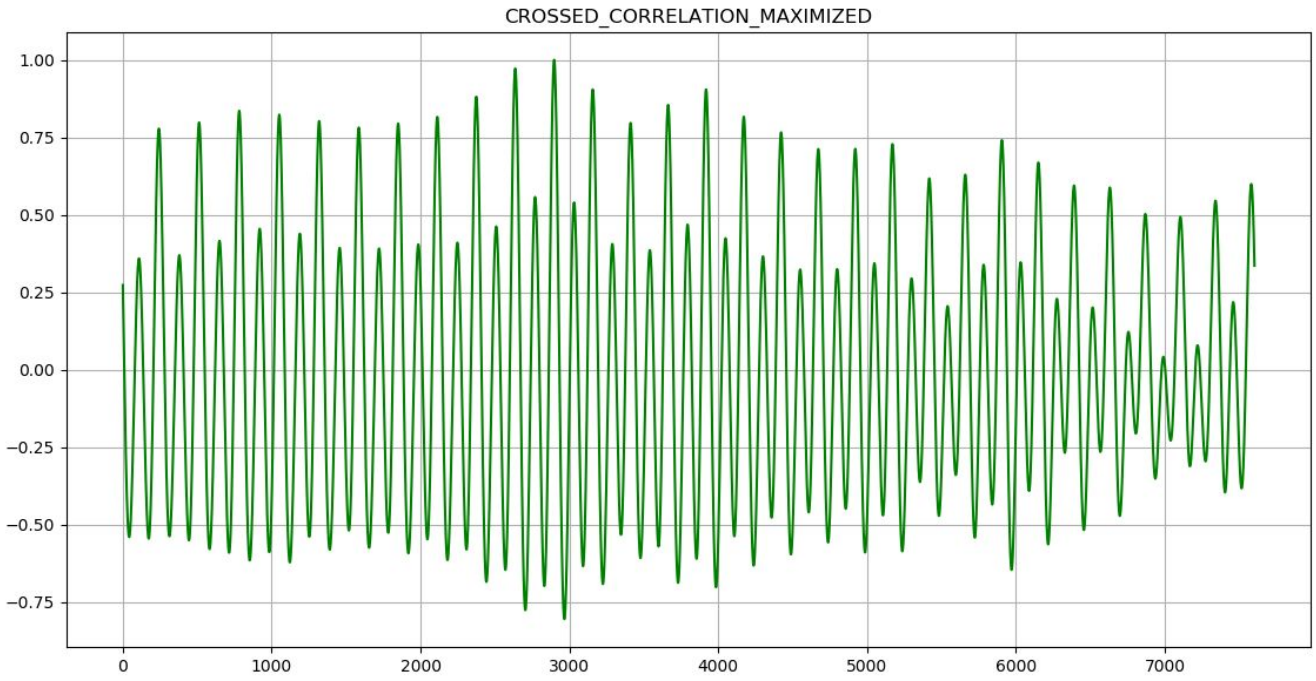

CROSSED_CORRELATION_NORMALISED
0.3
0.2
0.1
0.0
−0.1
−0.2
0
1000
2000
3000
4000
5000
6000
7000

**c.)** *Maximization:*

Maximization of the correlated data helps us in two aspects.First,it helps us in visualising the plotted data over a uniform scale so that the sample data of various wav files can be compared to draw a suitable conclusion.Second,it helps in getting a threshold value which is maintained in obtaining the dominant peaks for time period calculation.

```
begin:
        wave_maximization (window, template)
                r1 = list();
                r = wave_correlation(window , template)
                maximum_value=max(r)
                for i from 0 to (length of r)
                        n = r[i] / maximum_value          //obtaining the maximization values
                r1.append(n)
        return r1
        plt.plot(wave_maximization(window, template))
        plt.show()
end
```





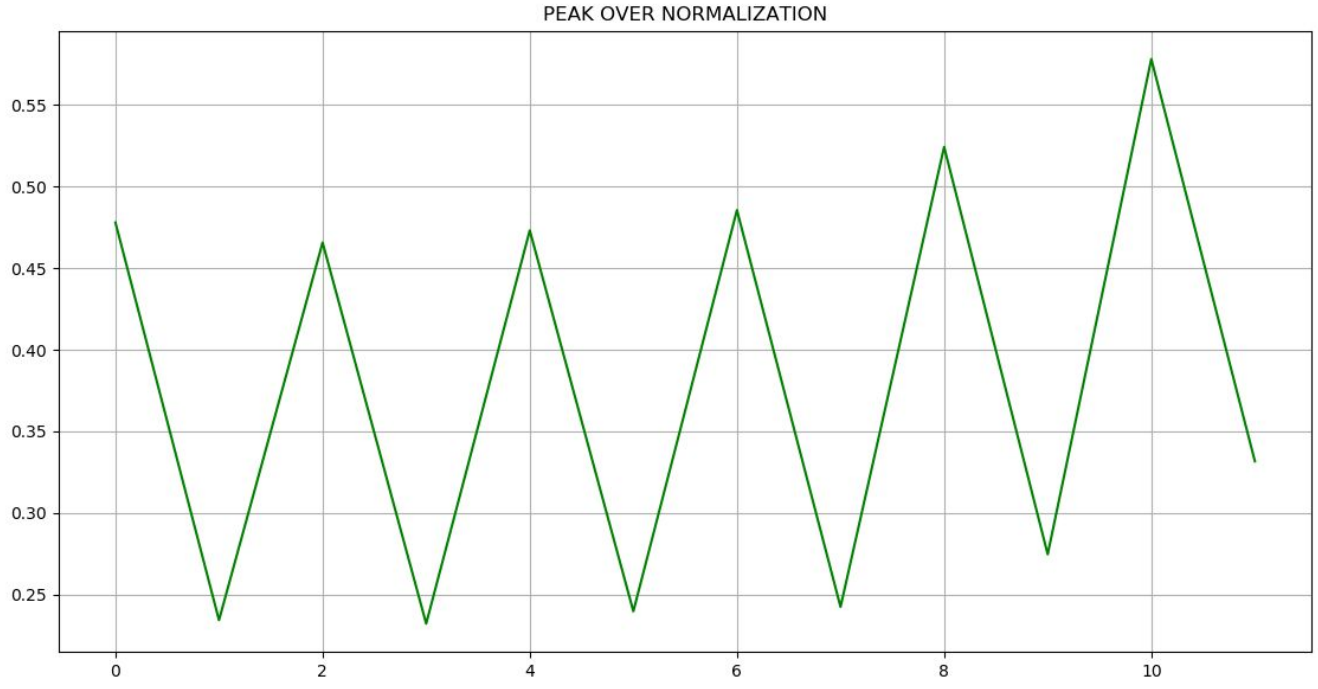

**d.)** *Finding the dominant peaks*:

After finding the normalised and the maximised waves over the correlated sample data, the *peak_finding* function is used to return the x-axis values and the y-axis values of the dominant peaks.

First, from maximised and normalised data set the dominant peaks are determined and plotted.This gives us a visualisation of where exactly the template have fitted into the window and a graphical value of ***time period*** of the given wav signal.The x-axis and the y-axis values are returned for theoretical verification.

```
begin:
        peak_finding(r):
                //x-axis array of the peaks
                peak_x_axis_values = signal.find_peaks_cwt(r, np.arange(1,2))
                peak = list();                                  //y-axis values of the peaks
                 n=0
                for i from 0 to (length of peak_x_axis_values)
                        n = r[peak_x_axis_values[i]]
                        peak.append(n)
                return peak , peak_x_axis_values
end
```

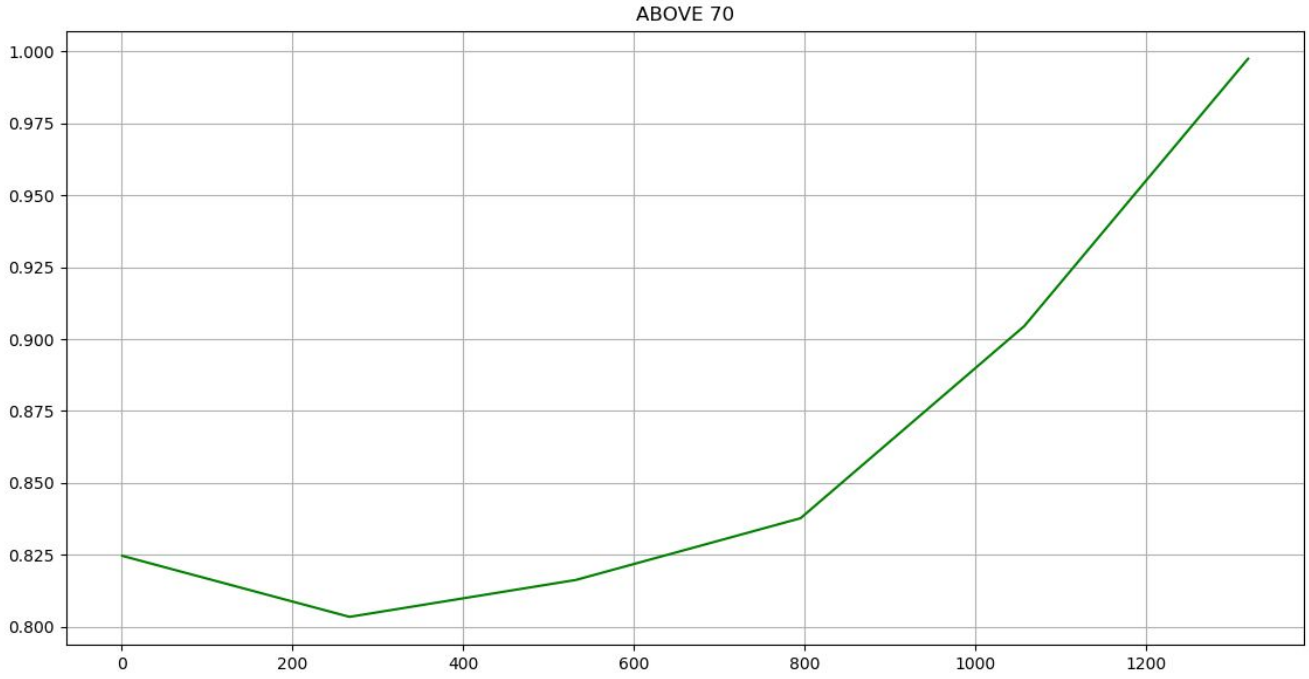
The *signal.find_peaks_cwt()* attempts to find the peaks in a 1-D array.The general approach is to smooth *vector* by convolving it with *wavelet(width)* for each width in *widths*. Relative maxima which appear at enough length scales, and with sufficiently high SNR, are accepted.



**e.)** *Filtering out the dominant peaks*:

To get more accurate peak values so that we can calculate the exact time period,we have filtered out the dominant peaks.A threshold value is set up and all the peaks below the threshold value are ignored.

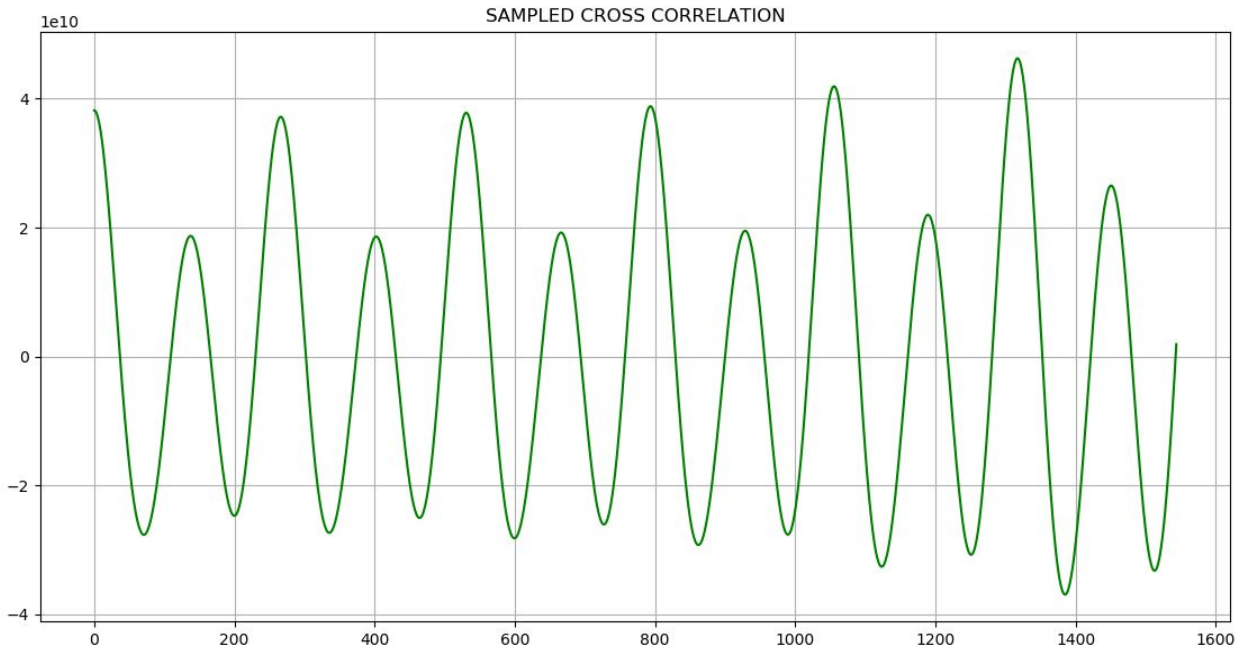
```
begin:
        remove_below70(x):
                y_axis,x_axis = peak_finding(x)
                 y_values = list();
                x_axis = list();
                for i from 0 to (length of y_axis)
                        if(y_axis[i]>0.74)
                                m=y_axis[i]
                                 n=x-axis[i]
                                y_values.append(m)
                                 x_axis.append(n)
                return y_values , x_values
        y_indice, x_indice=remove_below70(wave_maximization(window,template))
        plt.plot(x_indice, y_indice , "ABOVE 70",'g-')
end
```

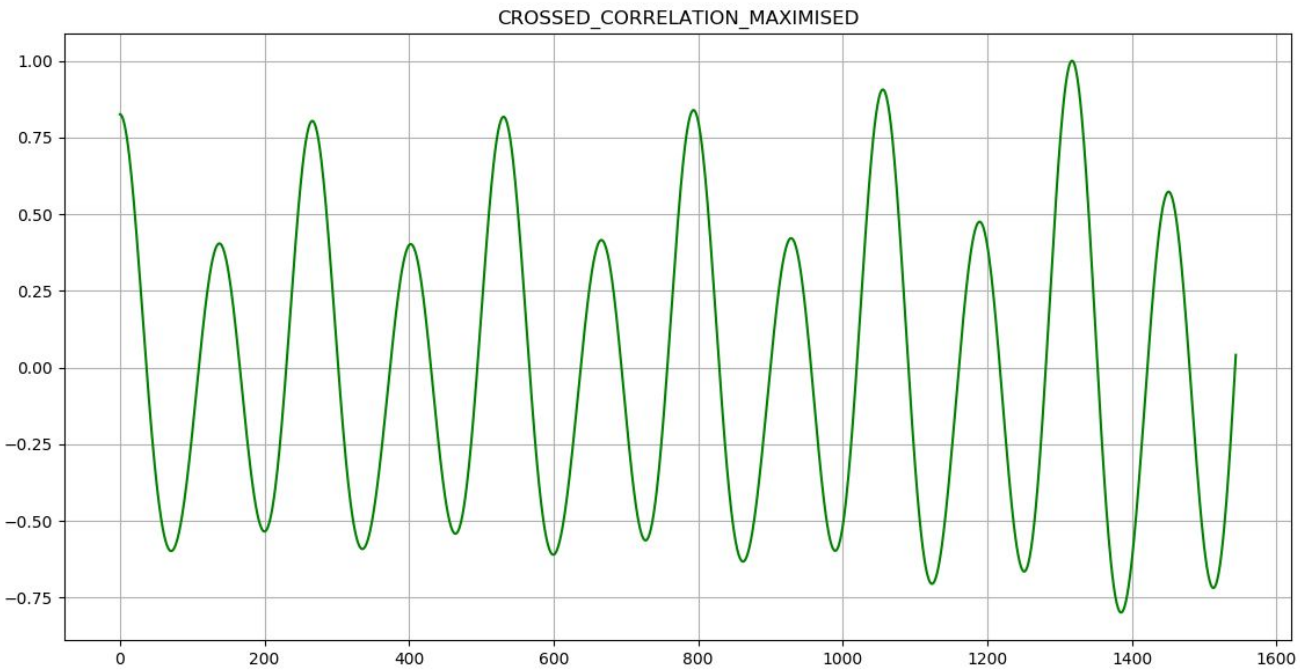


**f.)** *Shifting window and template creation:*

In order to get a more prominent and uniform data set we have divided the entire signal into windows of fixed length shifting with respect time.The template is also taken from the respective windows so that the amplitudes remain constant throughout when correlated.

```
begin:
        //Creating shifting window and template depending of the given time_slot
        window_creation(filtered_data, time_slot ,delay ,samplerate):
                sample = time_slot * samplerate
                r = list();
                for i from 0 to sample
                        n=filtered_data[i+delay]
                        r.append(n)
                return r
end
```






The first probable characteristic trait of an audio signal ***Time Period***

Depending upon all the above analysis we are able to finally calculate the time period of any given signal.

```
begin:
        shifting_template = any numeric value
        shifting_window = any numeric value
        sample_template= shifting_template * samplerate
        length = samples/sample_template
        delay = 0
        time_Period = list();
        for i from 0 to length
                dynamic_window = window_creation(filtered_data ,shifting_window ,delay ,samplerate)
                dynamic_template=window_creation(filtered_data, shifting_template ,delay, samplerate)
                y_indice, x_indice=remove_below70(wave_maximization(dynamic_window , dynamic_template))
                n = (x_indice[1]-x_indice[0])/samplerate
                time_Period.append(n)
                delay = x_indice[-1]

        print(time_Period)
end
```

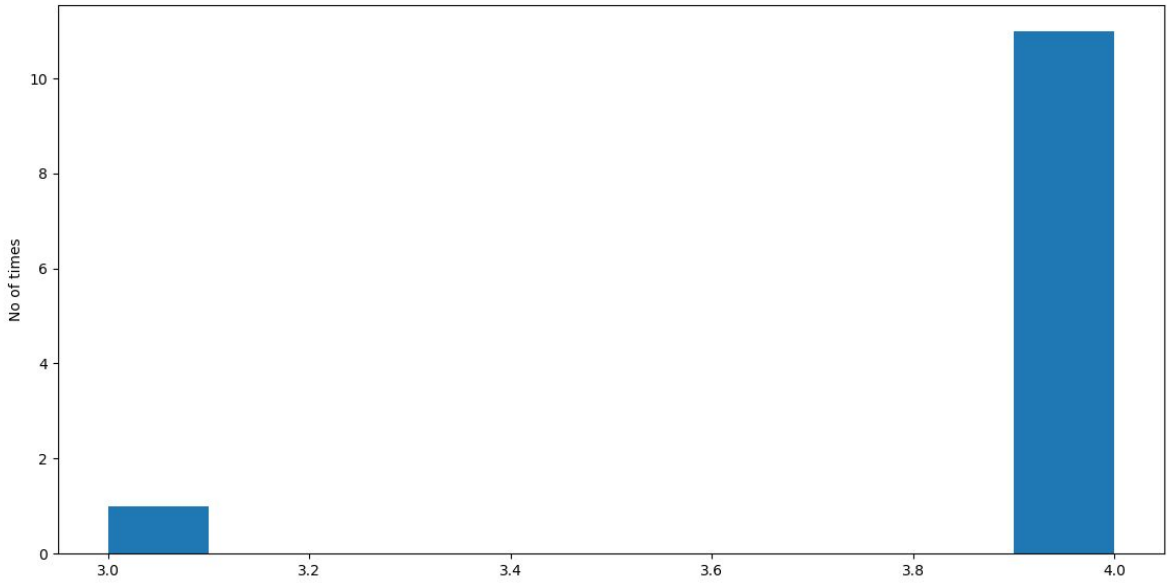
```
Time period= [0.006122448979591836, 0.006077097505668934, 0.006031746031746032, 0.006031746031746032,
0.006031746031746032, 0.006031746031746032, 0.006031746031746032, 0.006031746031746032, 0.006031746031746032,
0.006031746031746032, 0.006031746031746032, 0.006031746031746032]
Average time period =0.006031746031746032
```

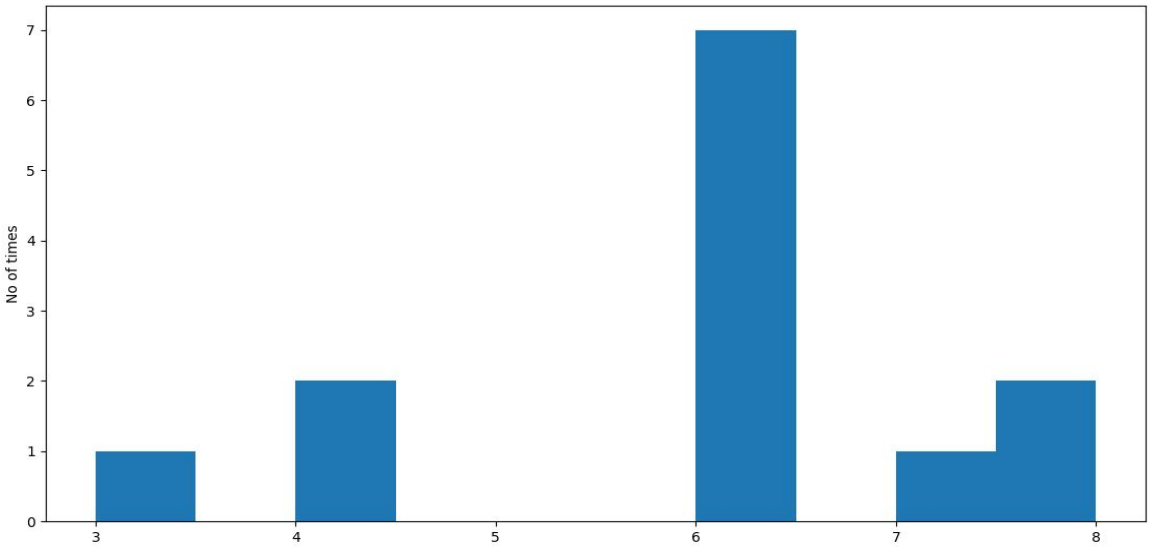
## 4. ***Zero Crossing Count(ZCC) over one period for different vowels***

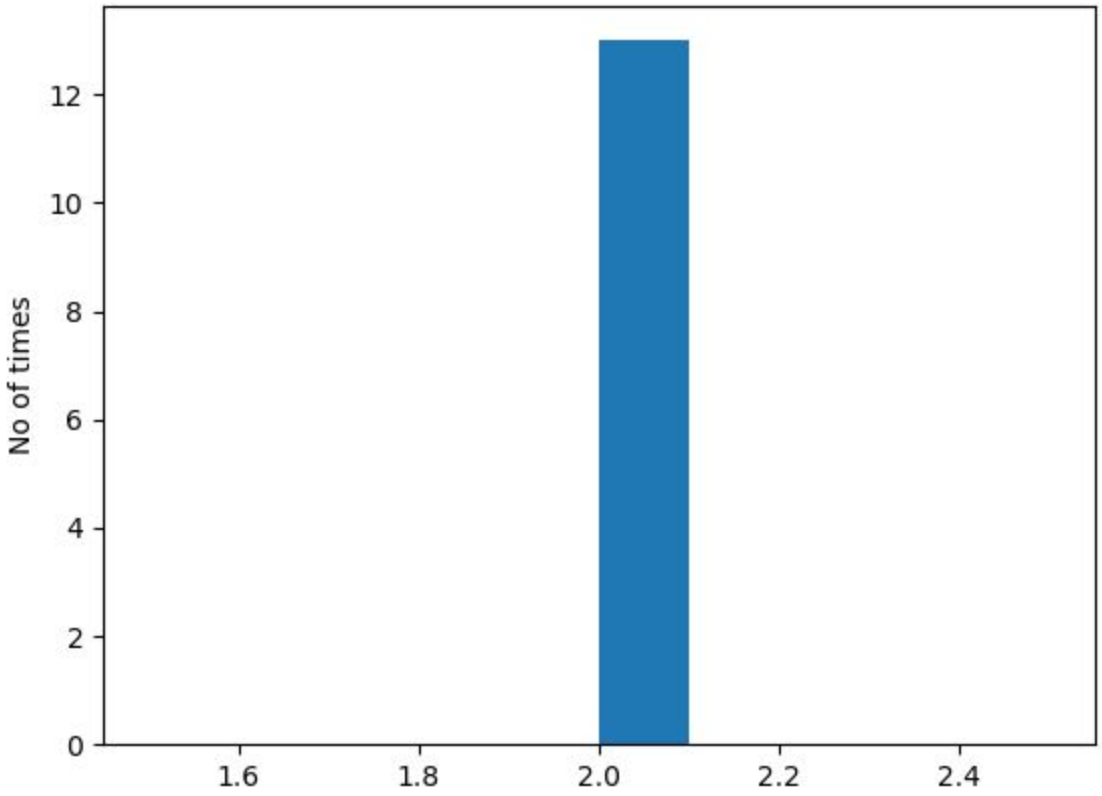
Heading towards observing the next probable characteristic trait of an audio signal *Zero Count*.

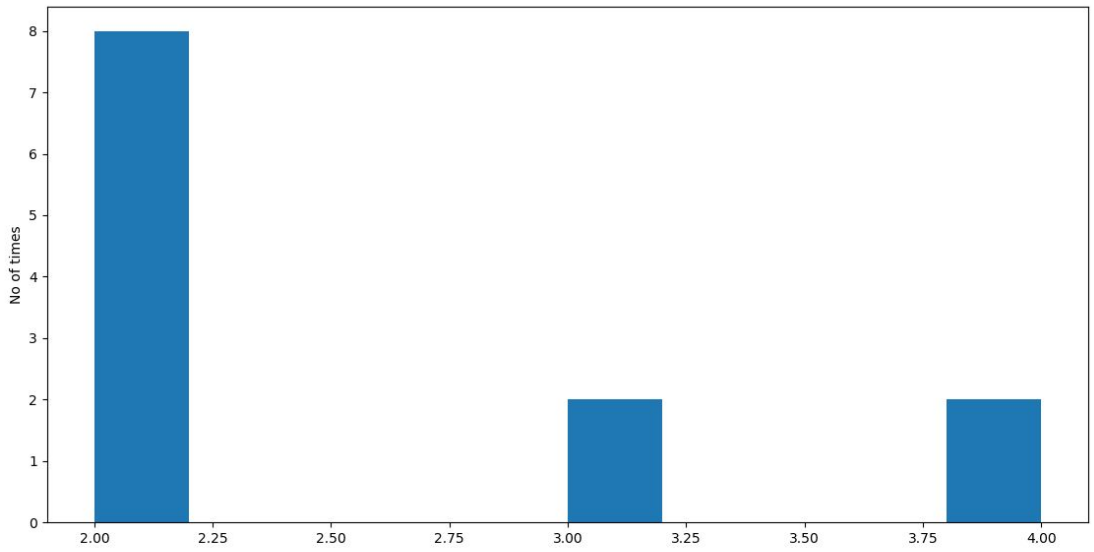
For each *Time period* , calculated above, we have determined the *Zero Count*. The average zero count of any signal is stored and analysed and displayed in the form of histogram.

```
begin:
        zero_crossing(x):
                z_count = 0
                for i from 1 to (length of x)
                        if(x[i-1]>0 and x[i]<0):
                                z_count = z_count + 1
                        elif(x[i-1]<0 and x[i]>0):
                                z_count = z_count + 1
                        else:
                                z_count  = z_count
                return z_count

        count = 0
        zerocount = list();
        for i from 0 to (length of time_Period)
                a = list();
                b = samplerate * time_Period[i]
                for j from 0 to b
                        m = filtered_data[j + count]
                        a.append(m)
                n=zero_crossing(a)
                zerocount.append(n)
                        count = count + b
        print(zerocount)
        plt.hist(zerocount)
end
```


No of times
0
2
4
6
8
10
3.0
3.2
3.4
3.6
3.8
4.0


No of times
0
1
2
3
4
5
6
7
3
4
5
6
7
8


No of times
0
2
4
6
8
10
12
1.6
1.8
2.0
2.2
2.4


No of times
0
1
2
3
4
5
6
7
8
2.00
2.25
2.50
2.75
3.00
3.25
3.50
3.75
4.00

## 5. ***Table for ZCC for different subjects for each vowel***

| | sub1 | sub2 | sub3 | sub4 | sub5 | sub6 | sub7 |
|---|---|---|---|---|---|---|---|
| e | 3,4 | 1,2,3,4 | 5,6 | 4,5,6 | 3,5,6 | 1,3,4 | 4 |
| i | 2,3,4 | 2 | 2 | 3,4,5 | 3,4,5 | 1,2,3 | 2 |
| o | 3,4,6,7,8 | 4,5,6 | 2,6 | 5,6 | 1,2,3,6 | 2,3,4 | 6 |
| u | 2 | 2 | 4 | 3,4,5 | 2,3,4,5 | 2 | 1,2,4 |

## 6. ***An overall comparison of the 4 vowels of different subjects.***

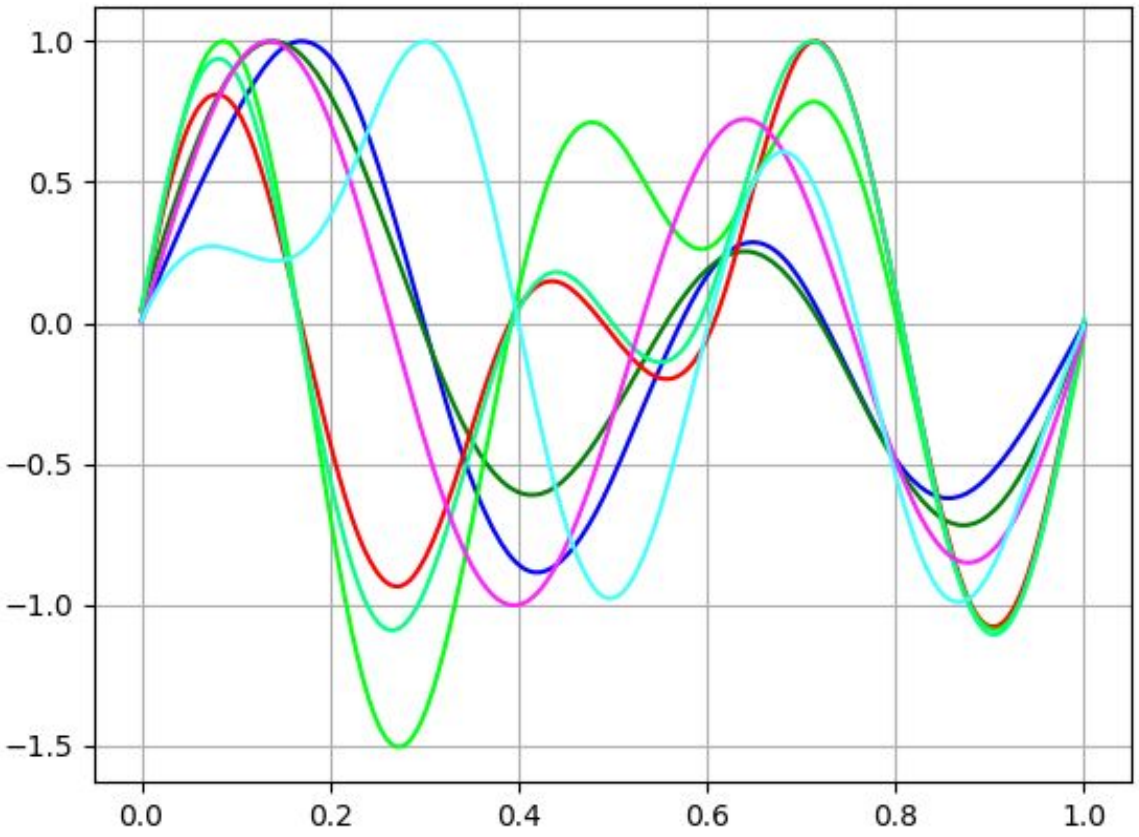
The signals are cut according to their respective time period and plotted.

A single plot contains the vowel waveform of different subjects maximized by itself.

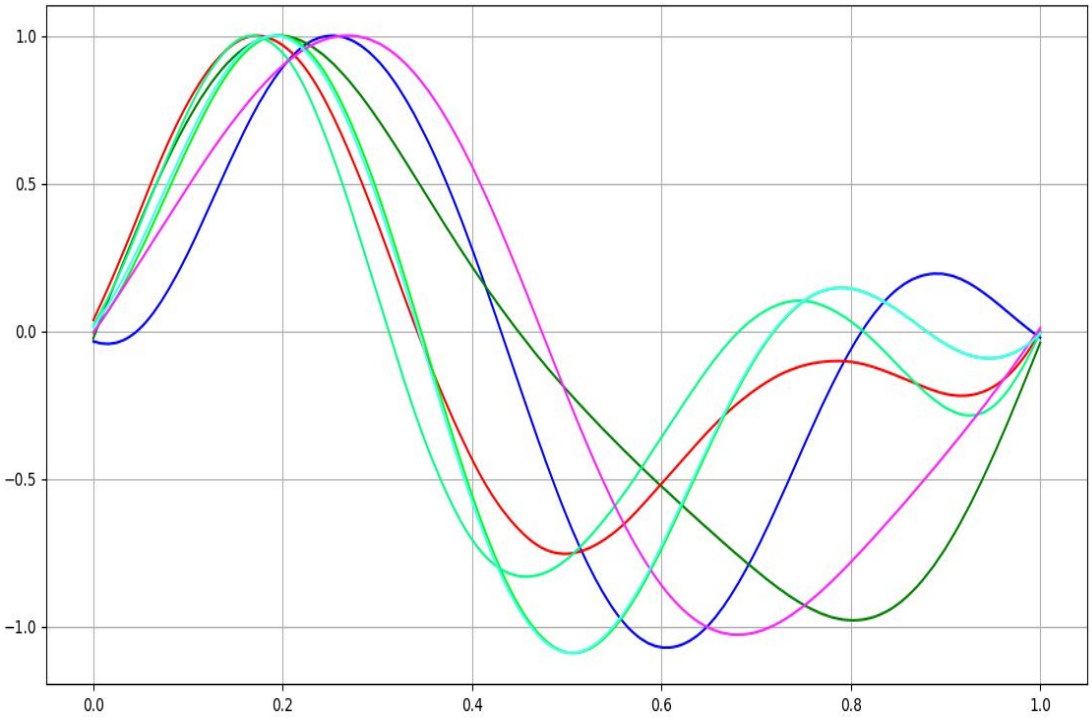
```
begin:
        name = input("enter  the name of the file ")
        name = name+".wav
        samplerate1, data1 = wavfile.read('file path'+name)
        data1 = data1[:,0]                          #real data
        samples1 = data1.shape[0]
        samplerate2, data2 = wavfile.read('file path '+name)
        data2 = data2[:,0]                          #real data
        samples2 = data2.shape[0]
        samplerate3, data3 = wavfile.read('file path '+name)
        data3 = data3[:,0]                          #real data
        samples3 = data3.shape[0]
        samplerate4, data4 = wavfile.read('file path '+name)
        data4 = data4[:,0]                          #real data
        samples4 = data4.shape[0]
        samplerate5, data5 = wavfile.read('file path '+name)
        data5 = data5[:,0]                          #real data
        samples5 = data5.shape[0]
        samplerate6, data6 = wavfile.read('file path '+name)
        data6 = data6[:,0]                          #real data
        samples6 = data6.shape[0]
        samplerate7, data7 = wavfile.read('file path '+name)
        data7 = data7[:,0]                          #real data
```

```
samples7 = data7.shape[0]

def wave_maximization(y):
r1=list();
maxm=max(y)
for i in range(len(y)):
        n=y[i]/maxm
        r1.append(n)
 return r1

plt.plot(np.linspace(0,1,samples1),wave_maximization(data1),'b-',
      np.linspace(0,  1, samples2),  wave_maximization(data2)  ,'g-',
      np.linspace(0,1,samples3),wave_maximization(data3),'r-',
      np.linspace(0,1,samples4),wave_maximization(data4),'#00FF12',
     np.linspace(0,1,samples5),wave_maximization(data5),'#05FE87',
     np.linspace(0,1,samples6),wave_maximization(data6),'#FF22FF',
     np.linspace(0,1,samples7),wave_maximization(data7),'#45FFFF',
     )
plt.show()
end
```

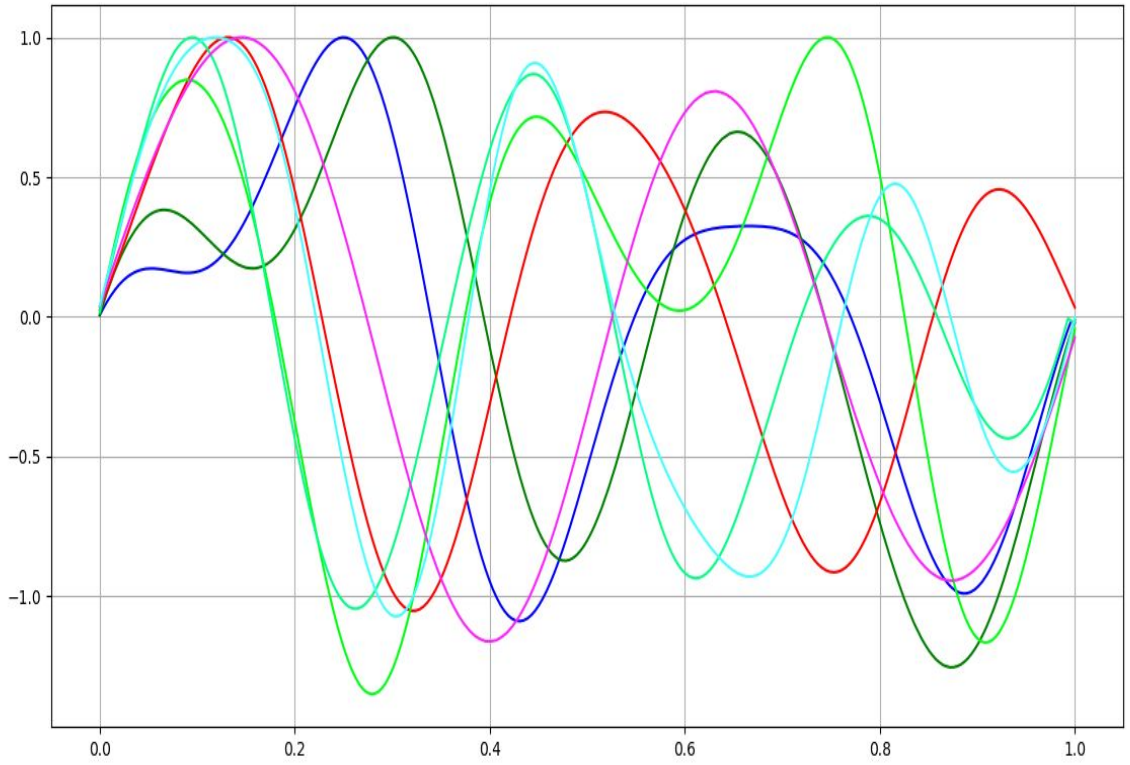
Subject comparison of vowel *a*



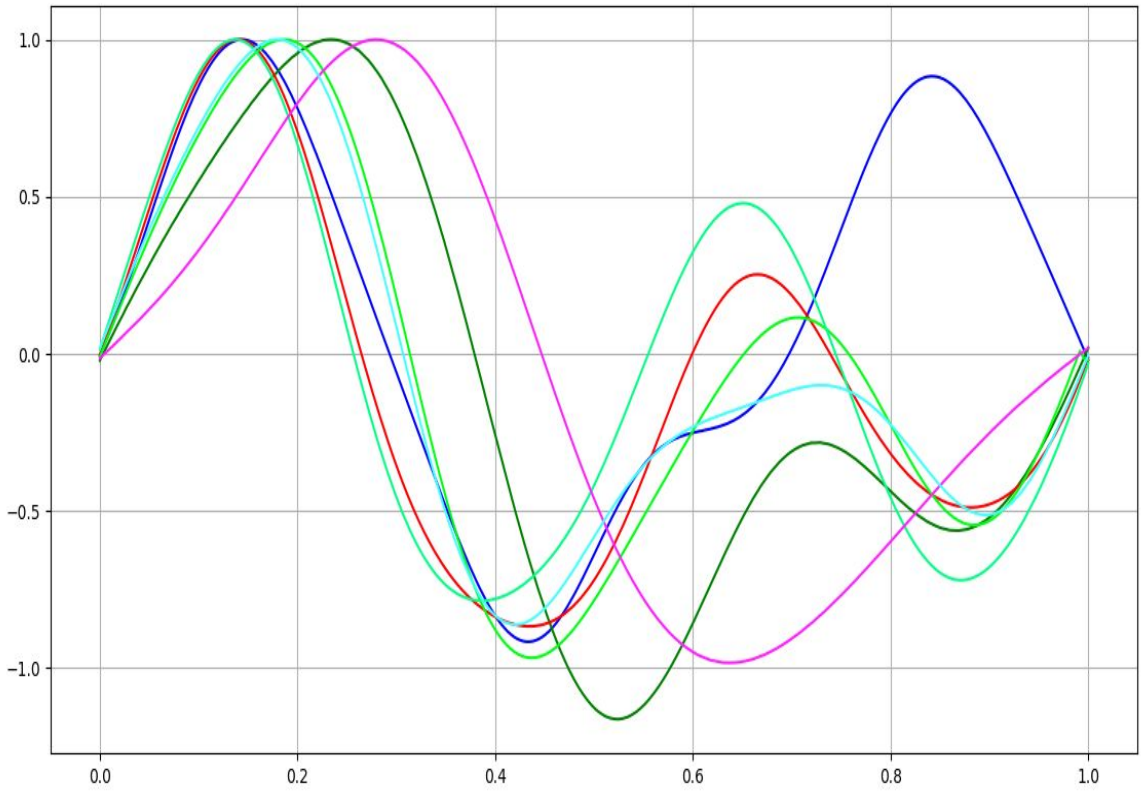
Subject comparison of vowel ***e***



Subject comparison of vowel ***o***



Subject comparison of vowel ***u***




## 7.***SYNTHESIS OF WAV FILE:***

Focussing over the time period and zero count trait of audio signal we have next tried to synthesise a wav file.

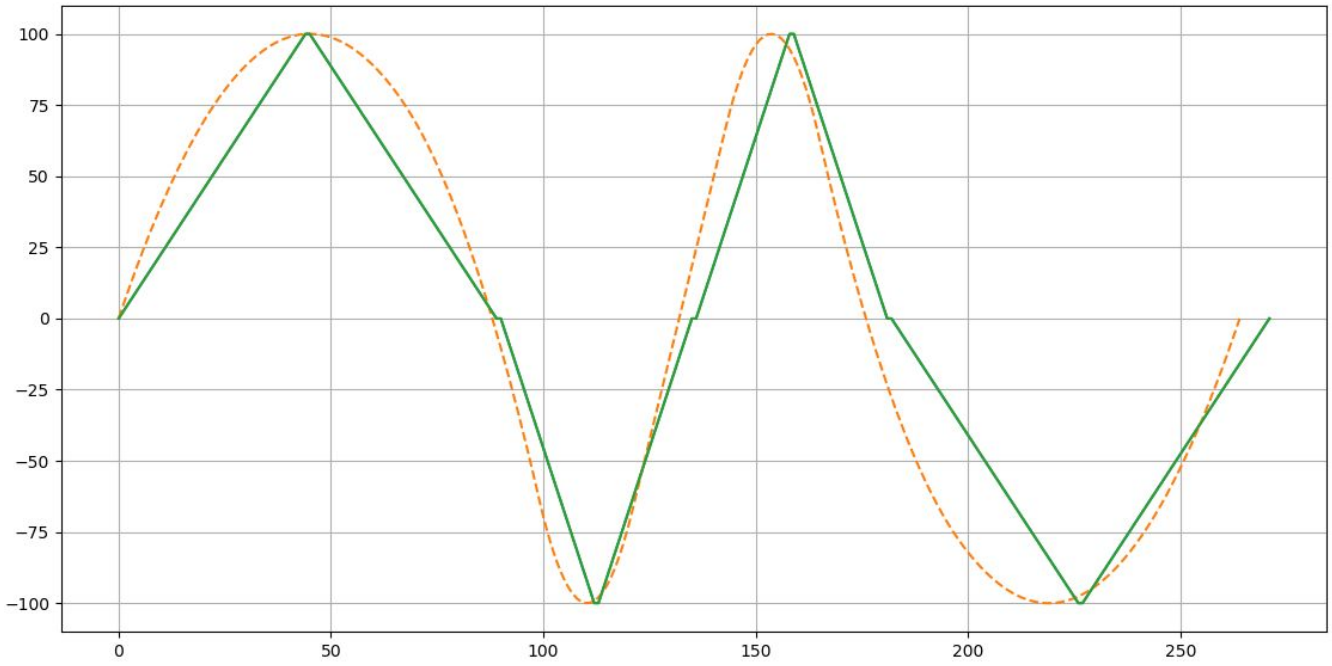
After writing a wav file using python and playing it we achieved an audio similar to the vowel pronunciation whose time period and zero count values are utilised in synthesizing the wave.

```
begin:
        Amplitude=100
        x = np.arange(0,  45)
        y = (Amplitude/44)*x
        x1=np.arange(44,89)
        y1=-(Amplitude/44)*(x-44)
        x2=np.arange(88,111)
        y2=-(Amplitude/22)*(x2-88)
        x3=np.arange(110,133)
        y3=(Amplitude/22)*(x3-110)-Amplitude
        x4=np.arange(132,155)
        y4=(Amplitude/22)*(x4-132)
        x5=np.arange(154,177)
        y5=-(Amplitude/22)*(x5-154)+Amplitude
        x6=np.arange(176,221)
        y6=-(Amplitude/44)*(x6-176)
        x7=np.arange(220,265)
        y7=(Amplitude/44)*(x7-220)-Amplitude
        spline = list();
        sharp_edge=np.array([])
        sharp_edge=np.append(sharp_edge,y)
        spline.append(sharp_edge[0])
```

```
            spline.append(sharp_edge[-1])

            sharp_edge=np.append(sharp_edge,y1)
            spline.append(sharp_edge[-1])
            sharp_edge=np.append(sharp_edge,y2)
            spline.append(sharp_edge[-1])
            sharp_edge=np.append(sharp_edge,y3)
            spline.append(sharp_edge[-1])
            sharp_edge=np.append(sharp_edge,y4)
            spline.append(sharp_edge[-1])
            sharp_edge=np.append(sharp_edge,y5)
            spline.append(sharp_edge[-1])
            sharp_edge=np.append(sharp_edge,y6)
            spline.append(sharp_edge[-1])
            sharp_edge=np.append(sharp_edge,y7)
            spline.append(sharp_edge[-1])
            for i in range(len(spline)):
                        print(spline[i])
            soft_edge = butter_lowpass_filter(sharp_edge, 1000, samplerate, order)
            f = interpolate.interp1d([0,44,88,110,132,154,176,220,264], spline, kind='quadratic')
            xnew = np.arange(0,265)
            ynew = f(xnew)
            plt.plot(np.arange(0,272),sharp_edge,'-',xnew,ynew,'--')
```

```
            plt.plot(sharp_edge)
            x_soft=np.arange(0,265)
            ynew=np.append(ynew,ynew)                    //Repeat it for a number of times and it will determine the length of the wav file
            wavfile.write("C:\\Users\\AILAB-4\\Desktop\\synthesis12.wav",samplerate,ynew)
            plt.plot(soft_edge)
            plt.show()
end
```

The **interp1d** class in **scipy.interpolate** is a convenient method to create a function based on fixed data points which can be evaluated anywhere within the domain defined by the given data using linear interpolation


100
75
50
25
0
−25
−50
−75
−100
0
50
100
150
200
250