Sound processing

Task 1

Fundamental frequency analysis

Jakub Janaszkiewicz

1. Table of Contents

Table of Contents

2. Introduction

Project consist creating an application enabling to read a *.wav file and to detect fundamental frequency in consecutive time windows. It is splitted in two parts. First consist code responsible for time domain method of fundamental frequency analysis and second one is for frequency domain methods. Both of the methods are working in the specified time windows.

3. Time domain method – Phase space analysis

3.1 Theoretical introduction

This method is based on cycles detection in a sound signal represented in a properly defined multidimensional (phase) space. The coordinates of following points at the pseudo-phase space are calculated in given order:

(x(i), y(i), z(i)) = (f(i+2*K), f(i+K), f(i))

This is three dimensional representation of signal delayed by multiples of K samples. In that project the K value has been setted to 10.

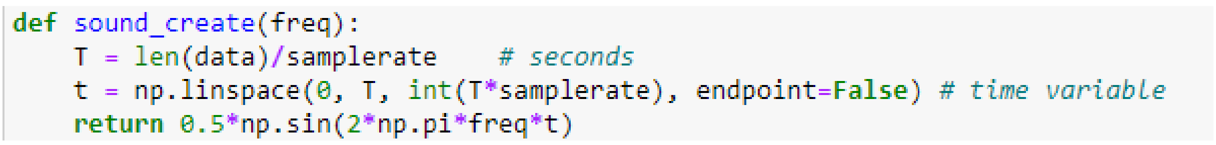
Prime period detection occurs when the signal coordinates reach or are close to reach the first coordinates. Distances between the points are calculated using the Euclidean distance calculation and the equation is as follows:

points_distance = ((xt[i]-xt[0])^2 + (yt[i]-yt[0])^2 + (zt[i]-zt[0])^2)^0.5

Thanks to this calculation we were able to measure the proper distance between points and to find the last coordinates of the period. If the starting point has been met by one of the points the fundamental frequency was revealed.

3.2 Implementation

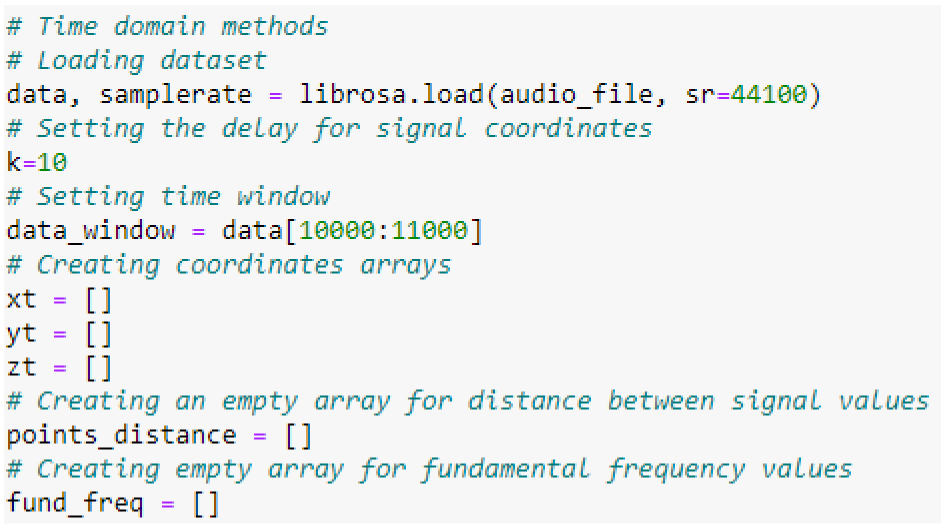
Code was implemented in python programming language with use of Jupyter Notebook environment.



```python
def sound_create(freq):
    T = len(data)/samplerate    # seconds
    t = np.linspace(0, T, int(T*samplerate), endpoint=False) # time variable
    return 0.5*np.sin(2*np.pi*freq*t)
```

Picture 1 Function implemented to create sinusoidal sound dataset

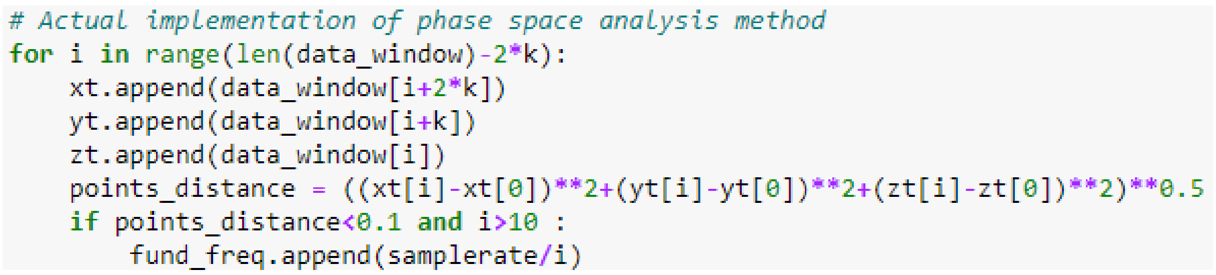
The function from Picture 1 is providing program with sinusoidal sound data with the length of the sound calculated in seconds. Using that function allows to create a sound with a passed fundamental frequency.



```python
# Time domain methods
# Loading dataset
data, samplerate = librosa.load(audio_file, sr=44100)
# Setting the delay for signal coordinates
k=10
# Setting time window
data_window = data[10000:11000]
# Creating coordinates arrays
xt = []
yt = []
zt = []
# Creating an empty array for distance between signal values
points_distance = []
# Creating empty array for fundamental frequency values
fund_freq = []
```

Picture 2 Loading dataset and preparing variables for phase space analysis

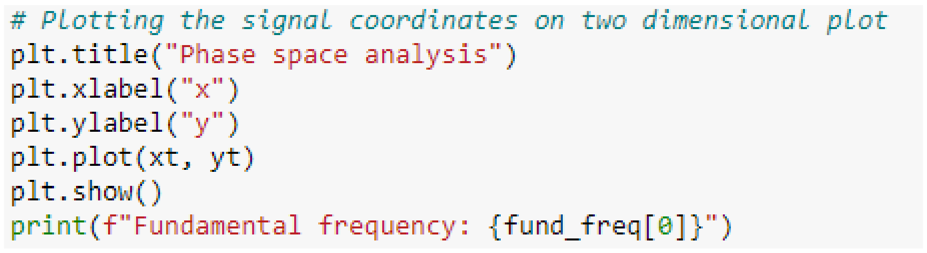
Picture 2 includes dataset loading, variables for the algorithm preparation, setting delay for signal coordinates value and length of data window separated from the whole dataset.



```python
# Actual implementation of phase space analysis method
for i in range(len(data_window)-2*k):
    xt.append(data_window[i+2*k])
    yt.append(data_window[i+k])
    zt.append(data_window[i])
    points_distance = ((xt[i]-xt[0])**2+(yt[i]-yt[0])**2+(zt[i]-zt[0])**2)**0.5
    if points_distance<0.1 and i>10 :
        fund_freq.append(samplerate/i)
```

Picture 3 Implementation of phase space analysis algorithm

In every iteration there are coordinates for signal created and the distance between every signal and first signal is calculated. If this signal is smaller then some given value then the fundamental frequency array is appended unless the number of signal is less then setted threshold (in this code it is 10 samples from start).



```
# Plotting the signal coordinates on two dimensional plot
plt.title("Phase space analysis")
plt.xlabel("x")
plt.ylabel("y")
plt.plot(xt, yt)
plt.show()
print(f"Fundamental frequency: {fund_freq[0]}")
```

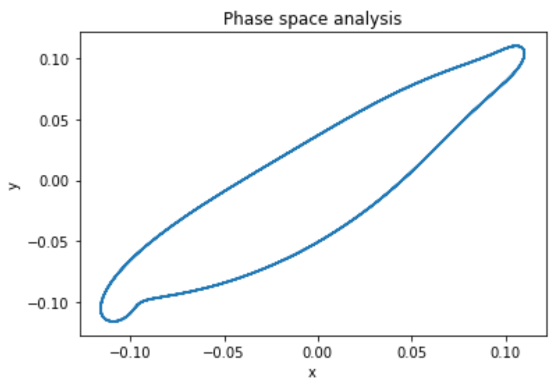
Picture 4 Plotting the trajectory of signals

Lines of code from Picture 4 are responsible for plotting the trajectory of signals in 2d plot.

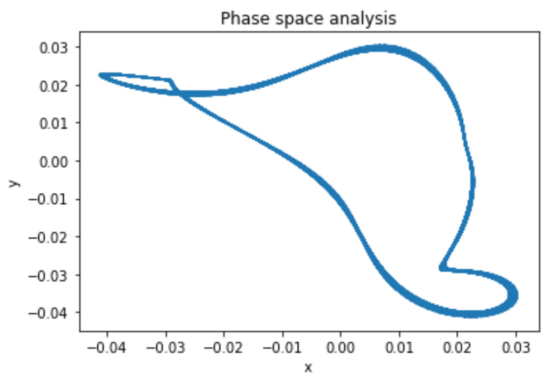
3.3 Results

The results were tested for different datasets. Examples a, b, c will contain variety of artificial generated sounds. D and E will contain natural sounds of flute and violin respectively. The range of sound types is from easy one tone examples to complicated sound datasets. Effectiveness of provided algorithm is tested on variety of examples. For proper working of algorithm values of threshold of distance between signals (distance_treshold) is changed for different datasets.

a) For easy tones of 225 Hz and 1708 Hz fundamental frequency where distance_treshold=0.003:

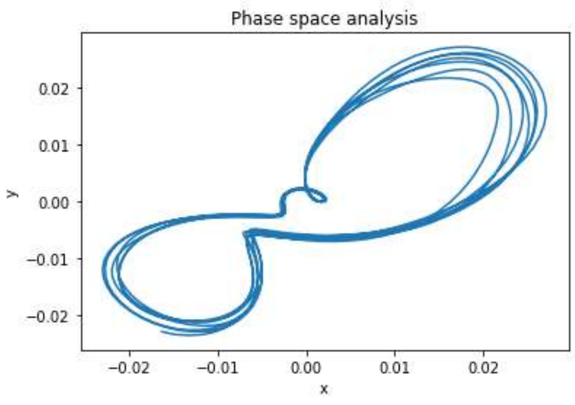


```
Fundamental frequency: 225.0
```

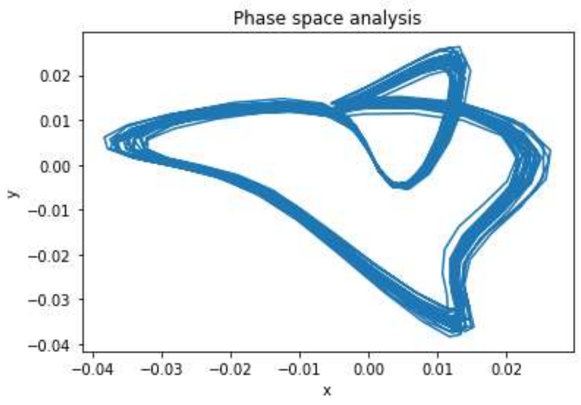


```
Fundamental frequency: 1696.1538461538462
```

b) For medium tones of 303 Hz and 1025 Hz fundamental frequency where distance_treshold=0.003:

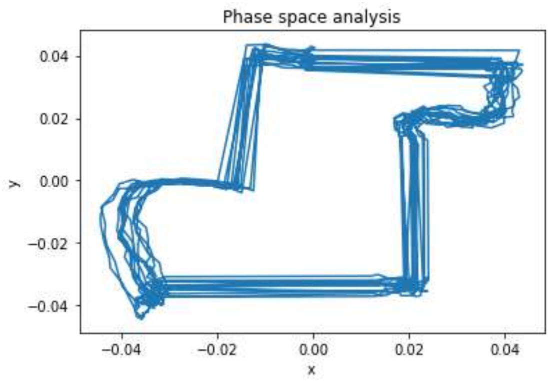


```
Fundamental frequency: 304.13793103448273
```

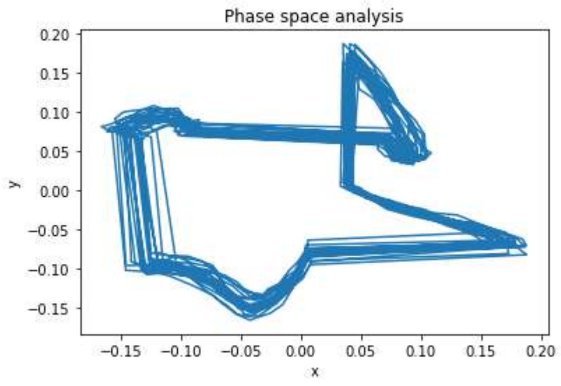


```
Fundamental frequency: 1025.5813953488373
```

c) For diff tones of 607 Hz and 911 Hz fundamental frequency where distance_treshold=0.002/0.03:

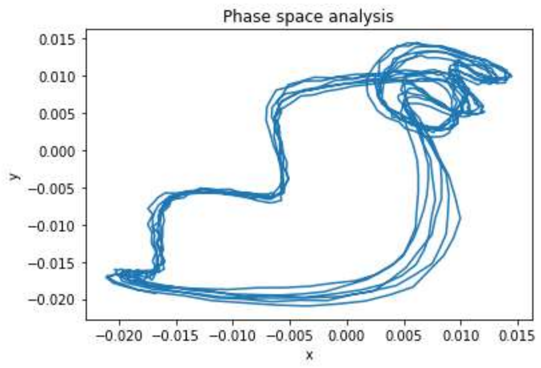


```
Fundamental frequency: 604.1095890410959
```

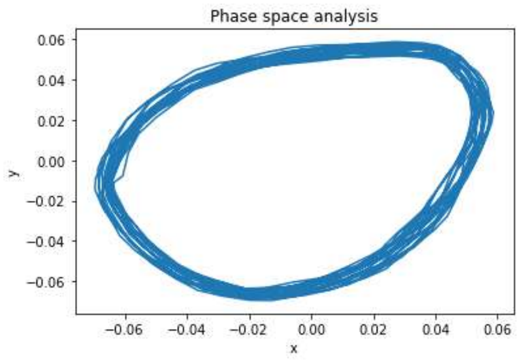


```
Fundamental frequency: 918.75
```

d) Natural flute sound with frequency of 276 Hz and 887 Hz where distance_treshold=0.003:

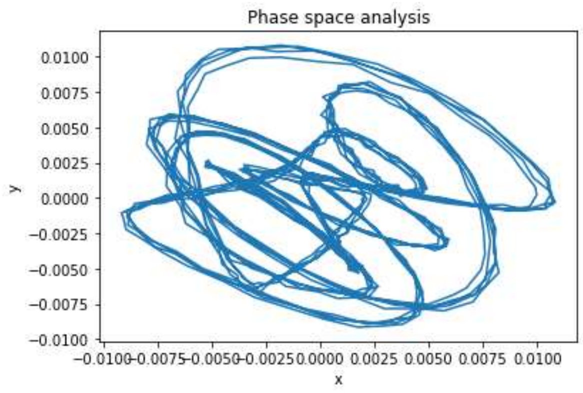


```
Fundamental frequency: 277.35849056603774
```

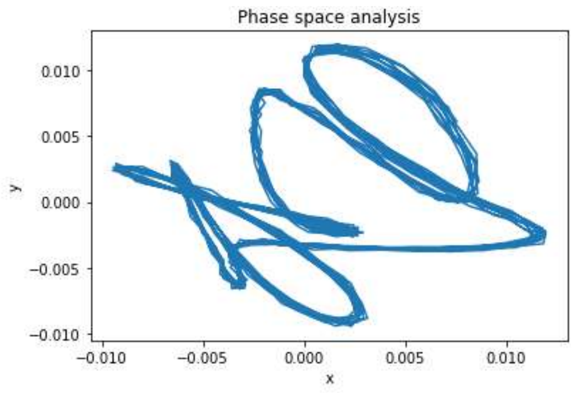


```
Fundamental frequency: 882.0
```

e) Natural violin sound with frequency of 196 Hz and 460 Hz where distance_treshold=0.003:



```
Fundamental frequency: 197.7578475336323
```



```
Fundamental frequency: 450.0
```

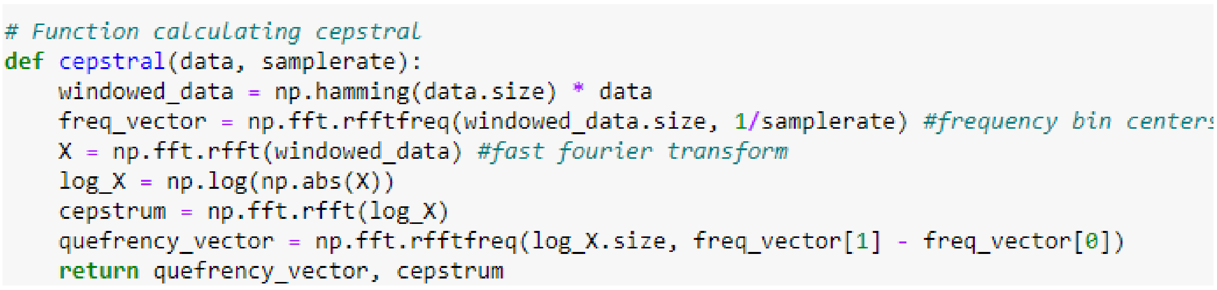
4. Frequency domain method – Cepstral analysis

4.1 Theoretical introduction

Cepstral analysis of sound data relies on computing Fourier magnitude spectrum on the base data with discrete Fourier transform. That would create a spectrum with definite pitch. After obtaining the magnitude spectrum of the input signal, its logarithm is computed before the second discrete Fourier transform. Magnitude spectrum of a magnitude spectrum is called Cepstrum. The fundamental frequency is then determined by the first valid pitch tracking.

4.2 Implementation of solution

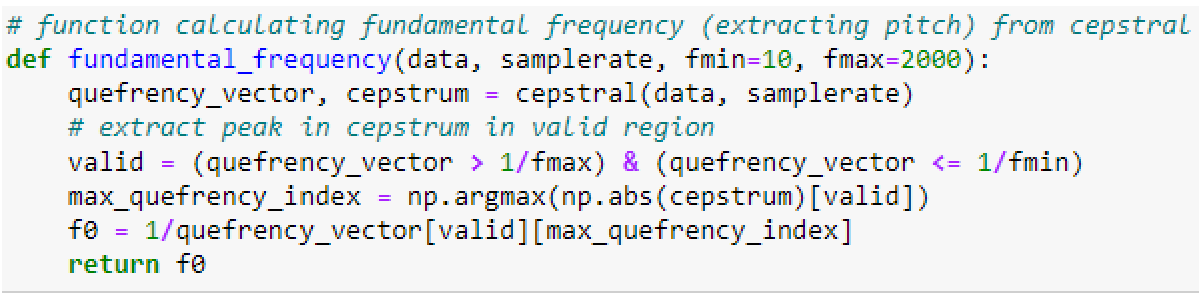
Code was implemented in python programming language with use of Jupyter Notebook environment.



```python
# Function calculating cepstral
def cepstral(data, samplerate):
    windowed_data = np.hamming(data.size) * data
    freq_vector = np.fft.rfftfreq(windowed_data.size, 1/samplerate) #frequency bin centers
    X = np.fft.rfft(windowed_data) #fast fourier transform
    log_X = np.log(np.abs(X))
    cepstrum = np.fft.rfft(log_X)
    quefrency_vector = np.fft.rfftfreq(log_X.size, freq_vector[1] - freq_vector[0])
    return quefrency_vector, cepstrum
```

Picture 5 Cepstral calculating function

Implementation of cepstral calculating function started with windowing prepared data with hamming method. Then frequency vector is calculated and outcome is discrete Fourier transform sample frequencies vector. It contains frequency bin centers in cycles per unit of the sample spacing. Next the fast Fourier transform is calculated on windowed data. After obtaining the magnitude spectrum of the input signal, its logarithm is computed and then the second fast Fourier transform is calculated.



```python
# function calculating fundamental frequency (extracting pitch) from cepstral
def fundamental_frequency(data, samplerate, fmin=10, fmax=2000):
    quefrency_vector, cepstrum = cepstral(data, samplerate)
    # extract peak in cepstrum in valid region
    valid = (quefrency_vector > 1/fmax) & (quefrency_vector <= 1/fmin)
    max_quefrency_index = np.argmax(np.abs(cepstrum)[valid])
    f0 = 1/quefrency_vector[valid][max_quefrency_index]
    return f0
```

Picture 6 Fundamental frequency calculating function

On Picture 6 there is fundamental frequency calculating function representation. First the Cepstrum calculation is conducted. Then the valid region for peak extraction is setted. Then the fundamental frequency in the valid area is extracted and passed as an outcome of the function.

```python
# Data Loading and windowing
audio_file = './natural/flute/887Hz.wav'
data, samplerate = librosa.load(audio_file, sr=44100)
data = data[25000:27000]

quefrency_vector, cepstrum = cepstral(data, samplerate)
# Plotting cepstrum
plt.title("audio file data")
plt.xlabel("frequency [Hz]")
plt.ylabel("amplitude")
plt.plot(data)
plt.show()
fig, ax = plt.subplots()
ax.plot(quefrency_vector[10:], np.abs(cepstrum[10:]))
ax.set_xlabel('quefrency (s)')
ax.set_title('cepstrum')
plt.show()
# Initializing function calculating fundamental frequency
fund_freq = int(round(fundamental_frequency(data, samplerate)))
print(f"Fundamental frequency: {fund_freq} Hz")
```

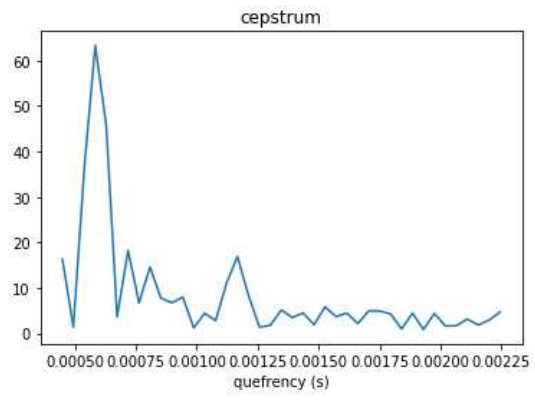
Picture 7 Dataset loading, plotting and calculating of Cepstrum and fundamental frequency

Code from Picture 7 is responsible from dataset loading and setting the data window for further analysis. Later the cepstral and fundamental frequency calculating functions are initialized. Last part is plotting cepstrum and printing extracted fundamental frequency value.

4.3 Results

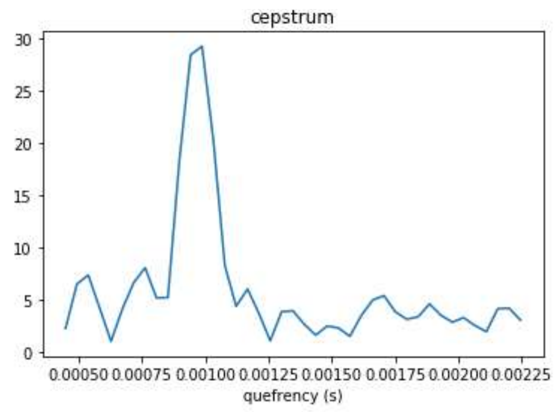
The results were tested for different datasets. Examples a, b, c will contain variety of artificial generated sounds. D and E will contain natural sounds of flute and violin respectively. The range of sound types is from easy one tone examples to complicated sound datasets. Effectiveness of provided algorithm is tested on variety of examples. For proper working of algorithm window of dataset (dataset_window) is changed for different datasets.

a) For easy tone of 1708 Hz fundamental frequency where dataset_window = [25000:25200]:



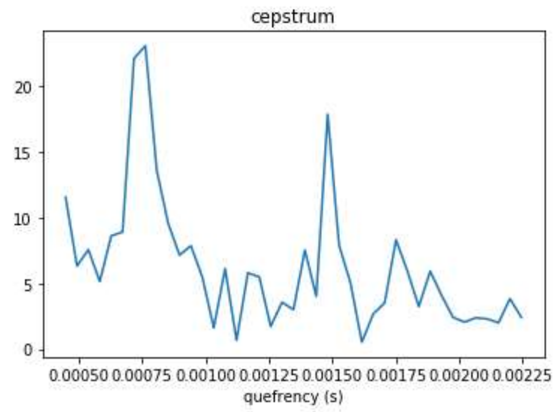
```
Fundamental frequency: 1713 Hz
```

b) For medium tone 1025 Hz fundamental frequency where dataset_window = [25000:25200]:



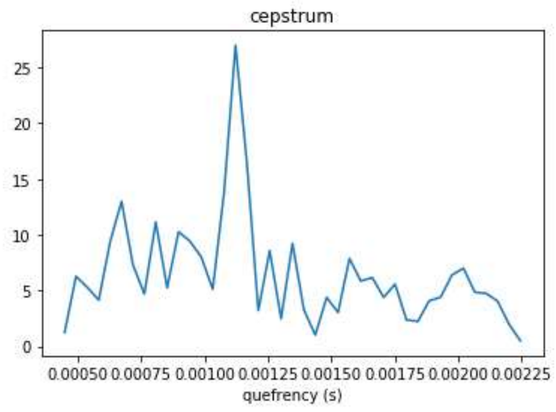
```
Fundamental frequency: 1012 Hz
```

c) For diff tone of 1366 Hz fundamental frequency where dataset_window = [25000:25200]:



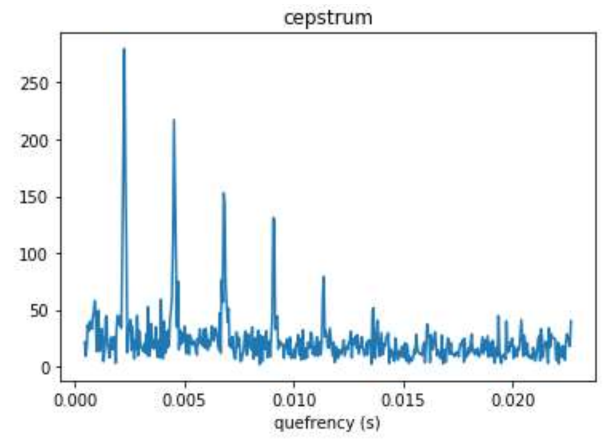
```
Fundamental frequency: 1310 Hz
```

d) Natural flute sound with frequency of 887 Hz where dataset_window = [25000:25200]:



```
Fundamental frequency: 891 Hz
```

e) Natural violin sound with frequency of 440 Hz where dataset_window = [25000:27000]:



```
Fundamental frequency: 441 Hz
```

5. Conclusions

For both algorithms the results obtained were really close to expected one. Obtained fundamental frequencies differentiate from actual one for the values of less than 20 Hz, so the algorithms are working efficiently. Implemented time domain method algorithm is giving slightly better and more valid outcomes. The necessity of changing threshold of distance between signals for phase space analysis and data windows for cepstral analysis are forcing user to analyse every given dataset individually. It prevents automatization of algorithms and decreases its usefulness. Nevertheless the algorithms are working properly and are giving valid results.