Novosibirsk State University
*THE REAL SCIENCE

Lab 2 - Signals

Department of Computer Science

Student: Patrushev Boris 19213

Advisor: Ruslan V. Akhpashev

Date: 19/02/2021

1. Form a chord / melody up to 5 seconds long (each one is unique)

First, let's import some libraries that will be useful to us in this lab:

- numpy - for working with numbers, arrays, mathematical operations
- matplotlib - for showing signal characteristics on the graph
- sounddevice - for playing signals(melody)

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
import sounddevice as sd
```

The first step is to generate and play an ordinary sine wave in audio. What is a sine wave? A sine wave(sinusoid) is an ordinary periodic function with certain properties. It looks like this:

$s(t) = A \cdot sin(f \cdot t + \phi)$,

where: *A* is the amplitude of the sinusoid (in most cases this is the signal power), *f* is the oscillation frequency, φ is the initial phase.

To generate a signal, it is necessary to determine in advance the parameters of the signal, which were described above.

- amplitude - the amplitude of the sinusoid
- duration [sec] - how long signal will play
- fs - time counts per second

np.arange *function generates an array of size* ***duration * fs*** *, then each element of the array is divided* ***fs****, getting small discrete time samples:)*

In [4]:
```
amplitude = 0.3
def timesamples(fs, duration):
    return (np.arange(np.ceil(duration * fs)) / fs)
```

- frequency[Hz]- the speed at which something repeats(signal frequency);
  I make a small function that converts a note name (from music) into frequency
  (more information here - https://en.wikipedia.org/wiki/Piano_key_frequencies):

In [5]:

```
def note(name):
    octave = int(name[-1])
    PITCHES = "c,c#,d,d#,e,f,f#,g,g#,a,a#,b".split(",")
    pitch = PITCHES.index(name[:-1].lower())
    return 440 * 2 ** ((octave - 4) + (pitch - 9) / 12)
```

And I decided to form a melody - hymn of Russia (USSR):

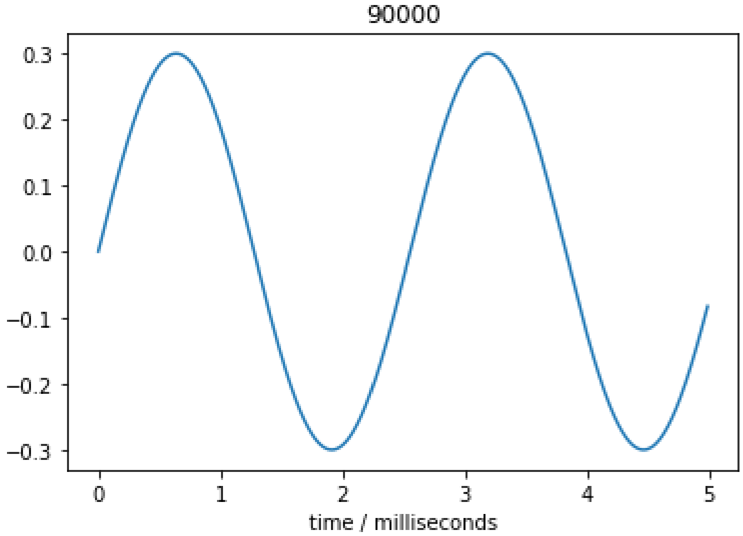
In [15]:

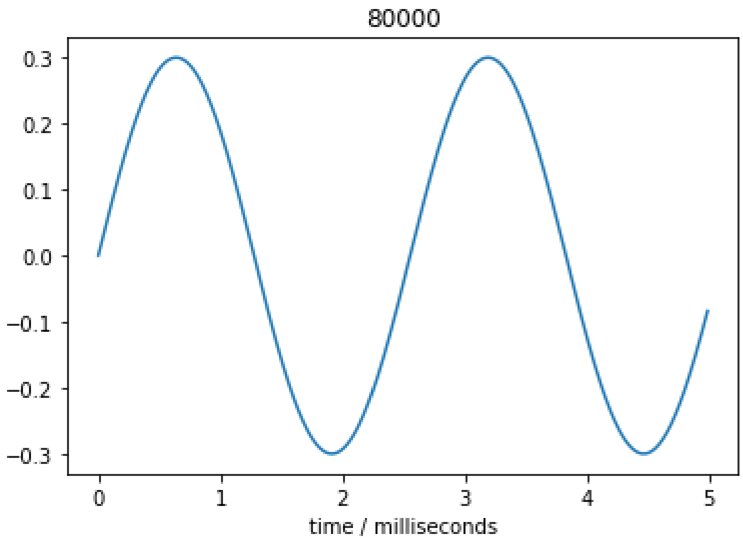
```
def melody(fs):
    sig1 = amplitude * np.sin(2 * np.pi * note("G4") * timesamples(fs, 1))
    sig2 = amplitude * np.sin(2 * np.pi * note("C5") * timesamples(fs, 1))
    sig3 = amplitude * np.sin(2 * np.pi * note("G4") * timesamples(fs, 1))
    sig4 = amplitude * np.sin(2 * np.pi * note("A4") * timesamples(fs, 0.25))
    sig5 = amplitude * np.sin(2 * np.pi * note("B4") * timesamples(fs, 0.75))
    return np.append(np.append(np.append(np.append(sig1, sig2), sig3), sig4), sig5)

hymn_ussr =melody(80000)
sd.play(hymn_ussr, 80000, blocking=True)
```

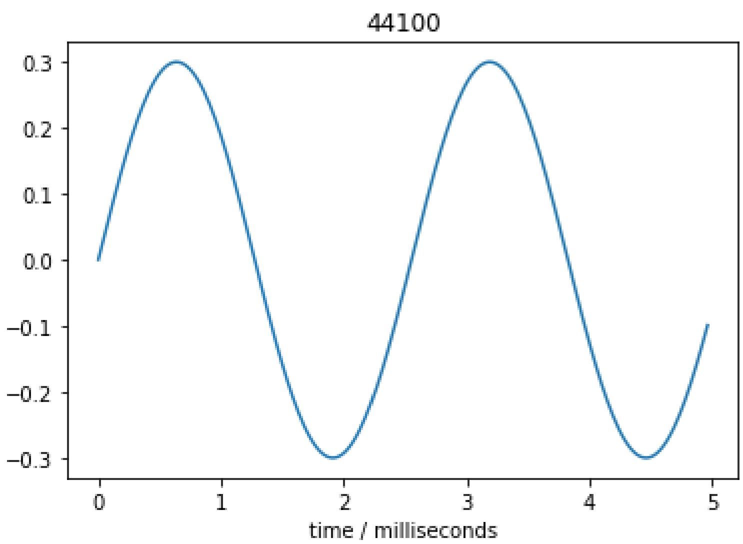
2-3. Check the effect of the sampling rate (1000; 50000; 5000 ++) on the quality of the melody being played. Draw conclusions(plots), describe; Display each option (at different sampling rates) as a function graph;

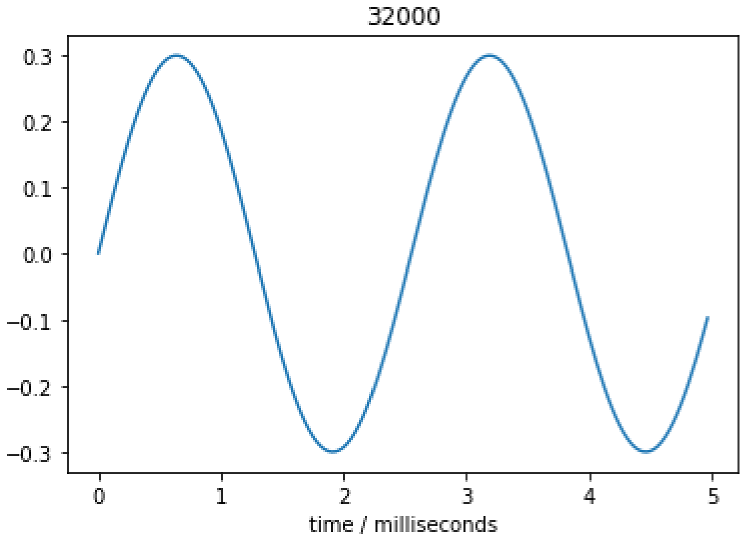
In [18]:

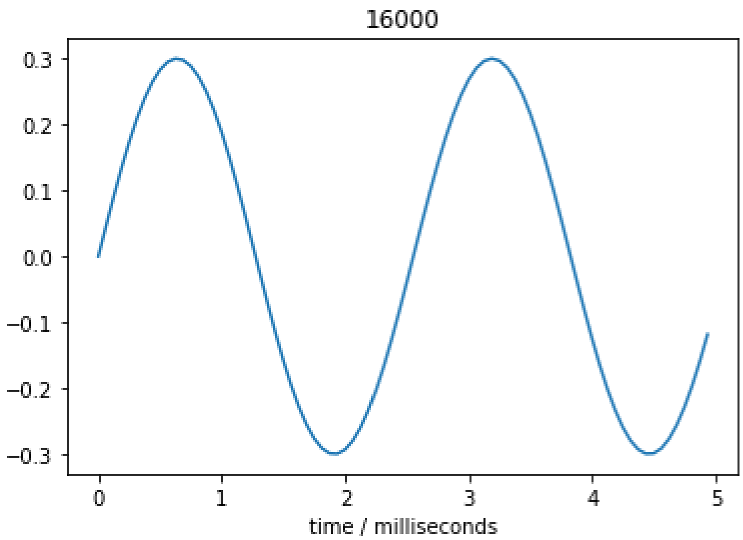
```
for i in [90000, 80000, 44100, 32000, 16000, 8000, 1000, 600]:
    hymn = melody(i)
    #sd.play(hymn, i, blocking=True)
    plt.plot(timesamples(i, 4)[:int(i / 200)] * 1000, hymn[:int(i / 200)])
    plt.title(i)
    plt.xlabel("time / milliseconds")
    plt.show()
```

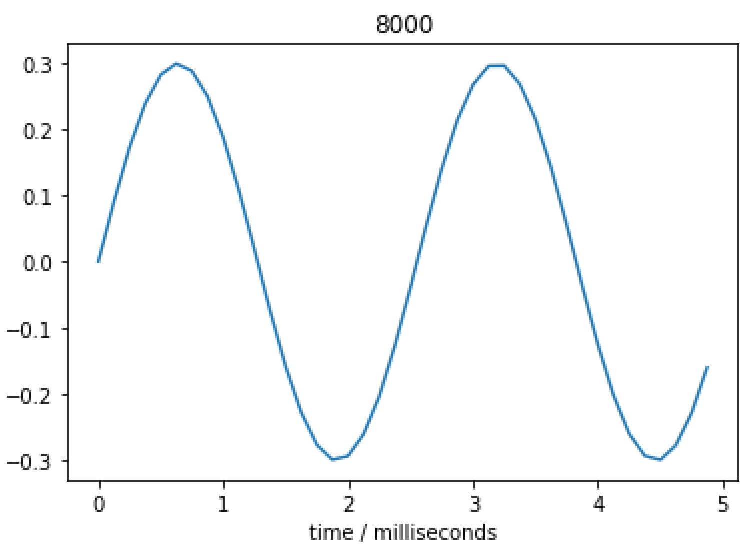

90000
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
1
2
3
4
5
time / milliseconds

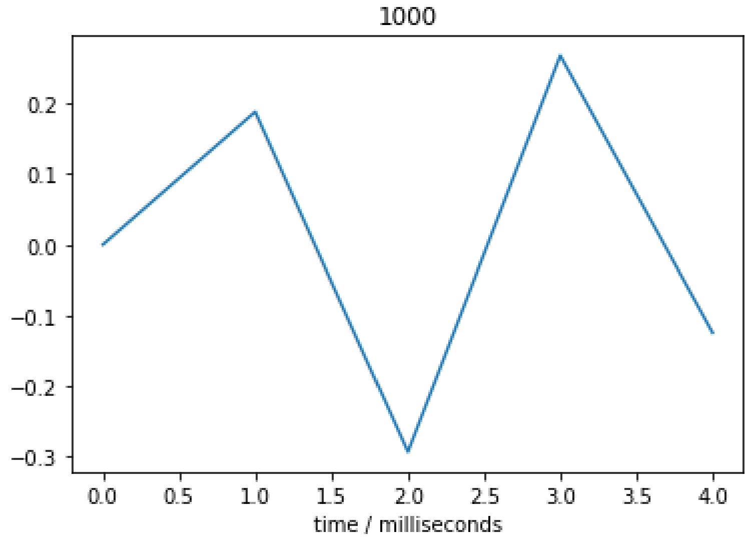

80000
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
1
2
3
4
5
time / milliseconds

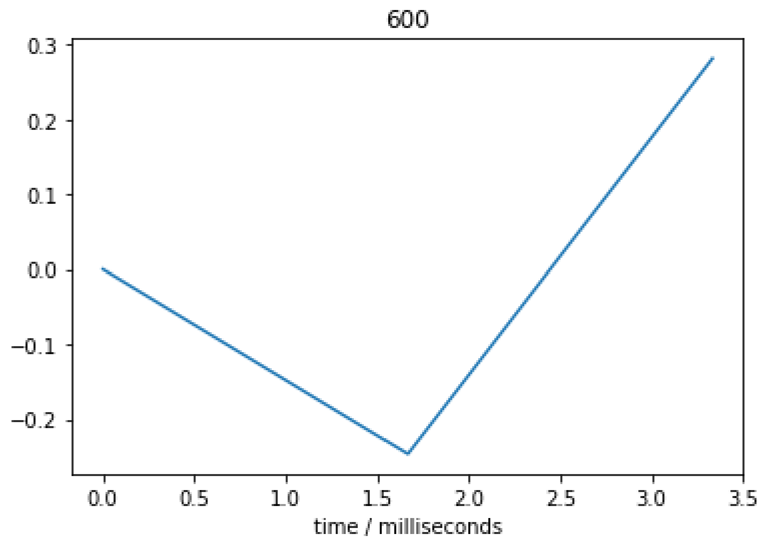

44100
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
1
2
3
4
5
time / milliseconds


32000
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
1
2
3
4
5
time / milliseconds


16000
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
1
2
3
4
5
time / milliseconds


8000
0.3
0.2
0.1
0.0
−0.1
−0.2
−0.3
0
1
2
3
4
5
time / milliseconds





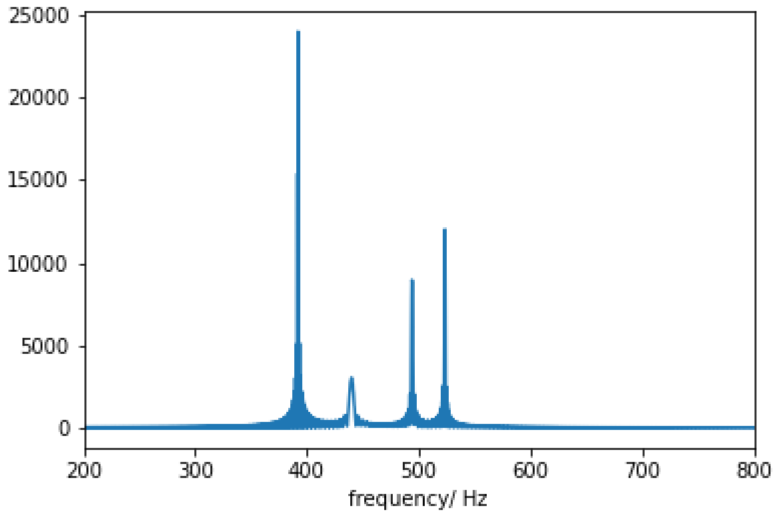
We can see and hear that at large values of **fs**, the changes are very small, which are not particularly noticeable to a person. But at values less than 10000, you can already see and hear that the quality is getting worse and worse. ("Rough music")

4. Study the principle and perform the Fourier transform (direct and inverse) for the original signal;

The *Fourier transform* is a tool that allows you to see the contribution of each of these harmonic components, characterized by a certain frequency, in the signal under investigation. In this sense, the Fourier transform is said to expand the function in terms of frequencies.

In [19]:

```
plt.plot(timesamples(40000, 4) * 40000/4, np.abs(np.fft.rfft(hymn_ussr))[0:-1])
plt.xlim(200, 800)
plt.xlabel("frequency/ Hz")
plt.show()
```



We can see on the graph that the leftmost frequency is G4(frequency about 391.9954)and then anothers

Now we can do the *inverse Fourier transform* and listen

In [12]:
```
sd.play(np.fft.irfft(np.fft.fft(hymn_ussr)), 80000, blocking=True)
```

So, we get hymn)