Санкт-Петербургский государственный политехнический
университет Петра Великого

**Высшая школа интеллектуальных систем и
суперкомпьютерных технологий**

Лабораторная работа

Дискретное косинусное преобразование

Работу выполнил студент
3-го курса, группа 3530901/80201
Сахибгареев Рамис Ринатович

Преподаватель:
Богач Наталья Владимировна

Санкт-Петербург 2021

Contents

List of Figures

Listings

1 Part 1: Amplitude effectiveness comparison

In this part we need to compare linalg and matrix ways of estimate amplitudes of DCT.

Firstly, lets define some helper function, analysis functions itself and analyzed wave.

```
    from thinkdsp import *
    from scipy.stats import linregress
    import thinkplot
    import numpy as np
    import matplotlib.pyplot as plt

    sig = UncorrelatedGaussianNoise()
    wave = sig.make_wave(duration = 1, framerate = 2**14)
    def plot_res(tm, ns, clr='cyan'):
        thinkplot.plot(ns, tm, color=clr)

        x = np.log(ns)
        y = np.log(tm)
        t = linregress(x,y)
        slope = t[0]

        return slope
    def analyze1(ys, fs, ts):
        args = np.outer(ts, fs)
        M = np.cos(PI2 * args)
        amps = np.linalg.solve(M, ys)
        return amps
    def analyze2(ys, fs, ts):
        args = np.outer(ts, fs)
        M = np.cos(PI2 * args)
        amps = np.dot(M, ys) / 2
        return amps

```

Listing 1: Functions definition

Now we can create a little benchmark, that performs analyze by incriminating length of the wave step by step.

```
    res1 = []
    res2 = []
    ns = 2 ** np.arange(6, 13)
    for n in ns:
        ts = (0.5 + np.arange(n)) / n
        fq = (0.5 + np.arange(n)) / 2
        ys = wave.ys[:n]
        r1 = %timeit -r1 -o analyze1(ys, fq, ts)
        r2 = %timeit -r1 -o analyze1(ys, fq, ts)
        res1.append(r1)
        res2.append(r2)

    t1 = [r.best for r in res1]
    t2 = [r.best for r in res2]
    print("first ", plot_res(t1, ns))
```

```
16      print("second ", plot_res(t2, ns, clr='orange'))
17
```

Listing 2: Benchmark code

```
first  1.6140712963703687
second  1.356972002159275
```

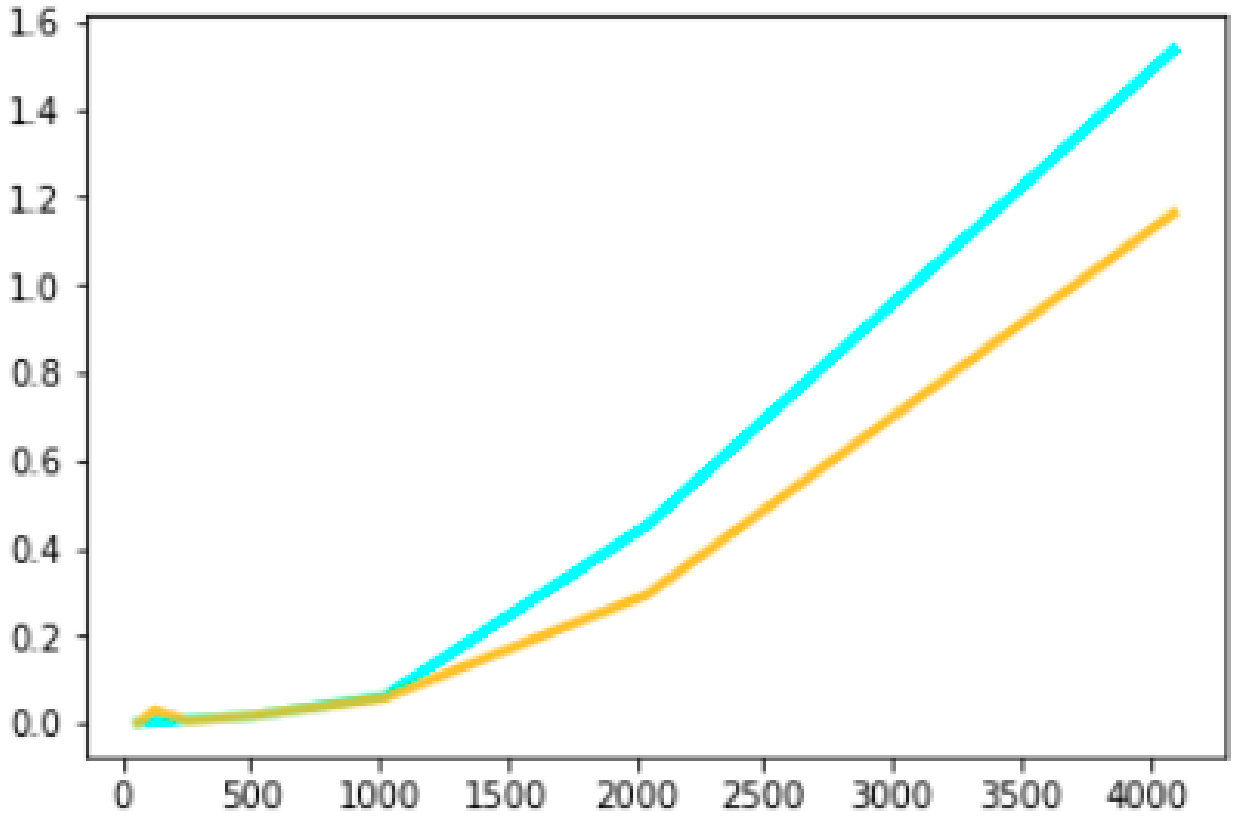

Figure 1: Benchmark result

We can see, that matrix way is much faster, than linear algebra way of estimating DCT amplitude.

2 Part 2: Sound compression algorithm

In this part we need to implement simple sound compression algorithm by splitting the signal into the parts and removing near to 0 components from spectrum. Let's define compression algorithm. Compress function removes near 0 components, `makedctspectrogra` makes DCT spectrogram.

```
    def compress(dct, thresh=1):
        count = 0
        for i, amp in enumerate(dct.amps):
            if np.abs(amp) < thresh:
                dct.hs[i] = 0
                count += 1

        n = len(dct.amps)
        return (count, n)
    def make_dct_spectrogram(wave, seg_length):
        window = np.hamming(seg_length)
        i, j = 0, seg_length
        step = seg_length // 2
        spec_map = {}
        while j < len(wave.ys):
            segment = wave.slice(i, j)
            segment.window(window)
            t = (segment.start + segment.end) / 2
            spec_map[t] = segment.make_dct()
            i += step
            j += step
        return Spectrogram(spec_map, seg_length)

```

Listing 3: Functions definition

Let's compress Halzion by YOASOBI. We can see. that its segment has a lot of near 0 components, that can be removed.

```
    wave = read_wave('sound/halzion.wav')
        segment = wave.segment(start=20, duration=0.1)
        segment.make_spectrum().plot()
        wave.make_audio()

```

Listing 4: Reading the waver

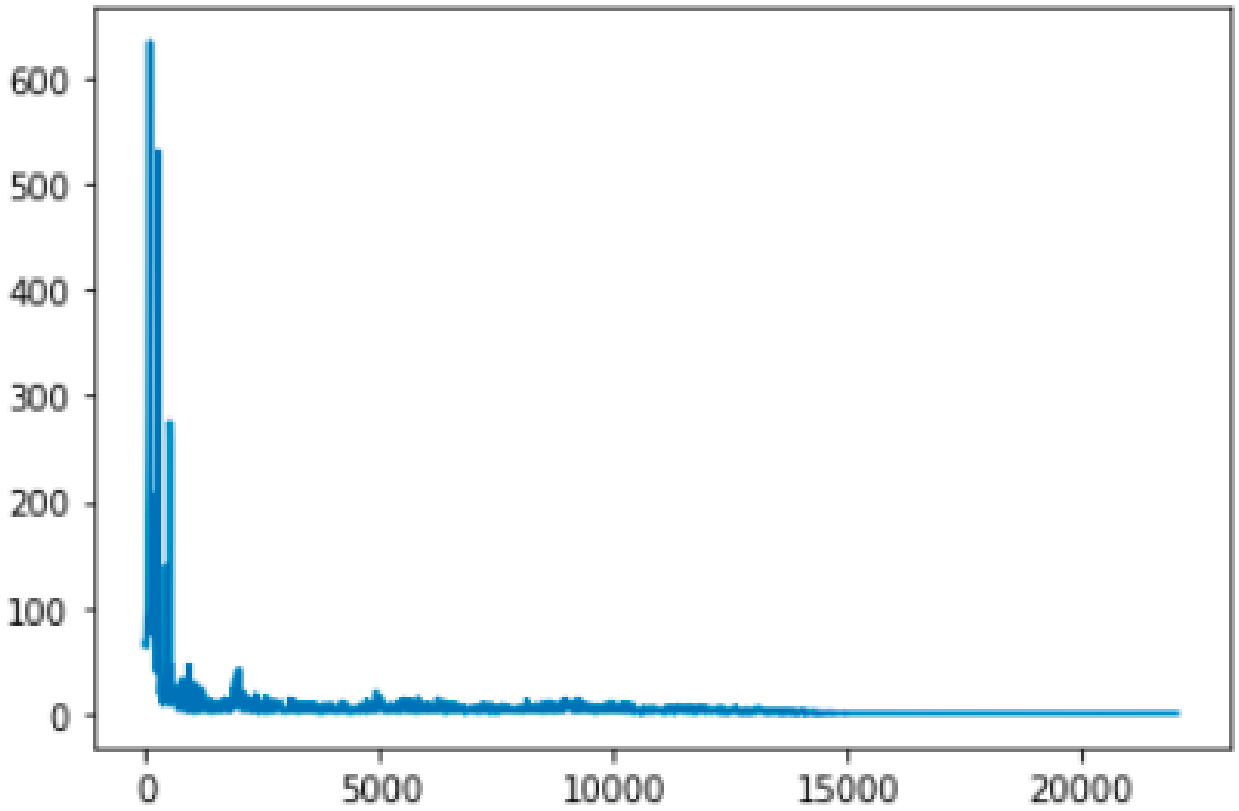

Figure 2: Halzion's segment spectrogram

By splitting this song into the segments of 512 elements and removing every component with amplitude less than a 0 we've got 67 percent spectrum size reduction.D

```
    spectrogram = make_dct_spectrogram(wave, seg_length=512)
    total = 0;
    rem = 0;
    for t, dct in spectrogram.spec_map.items():
        cnt, n = compress(dct, thresh=1)
        total += n;
        rem += cnt;
    print("removed", rem, "of", total, "percent", rem / total)

```

Listing 5: Compression of the wave

```
removed 11777322 of 17519616 percent 0.6722363092889707

In [20]: spectrogram.make_wave().make_audio()

Out[20]:
```

Figure 3: Halzion compression result

By listening the resulting audio I can say, that sound now is unpleasant to hear because of buzzing.

3 Part 3: Playing with a phace

In this part we need to investigate a phase effect on the signal perception.

To do it thiangle signal with frequency of 500 Hz was generated. If we will plot its phases plot we see, that we have a lot of noise on the plot because of aliasing effect.

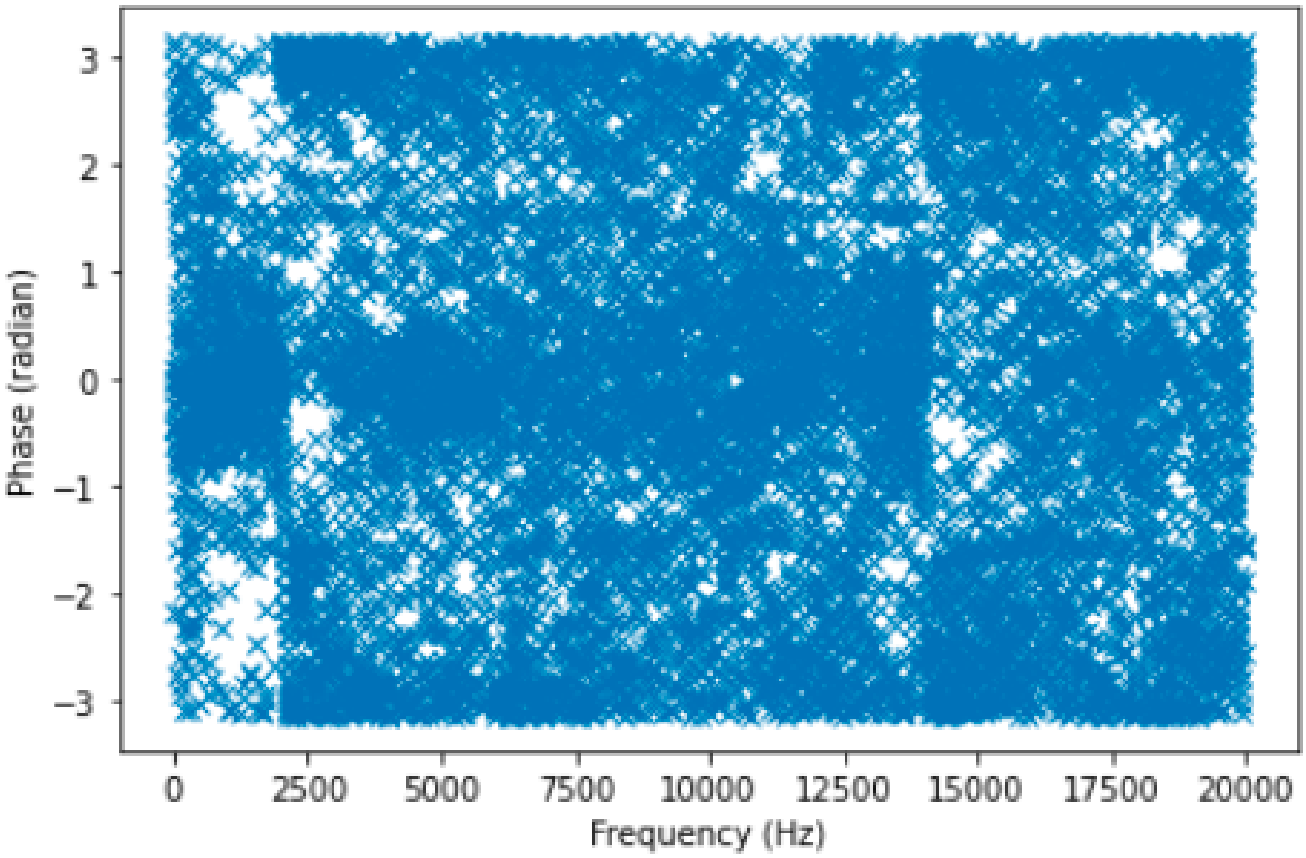

Figure 4: Phases noise

We can get rid of this noise by cutting off frequencies with amplitude lower than threshold. The resulting set of plots is next. We can see, that phase increases linearly by its frequency.

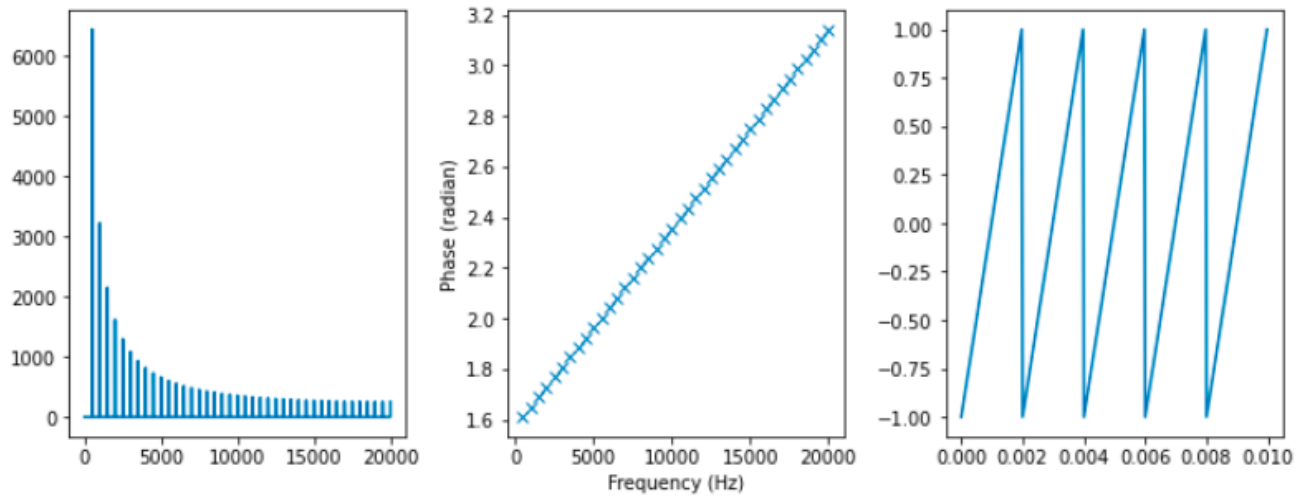

Figure 5: Triangle signal plot set

If we set phase of every frequency to 0, we see that signal changed dramatically, but we cannot hear any difference with the original one.

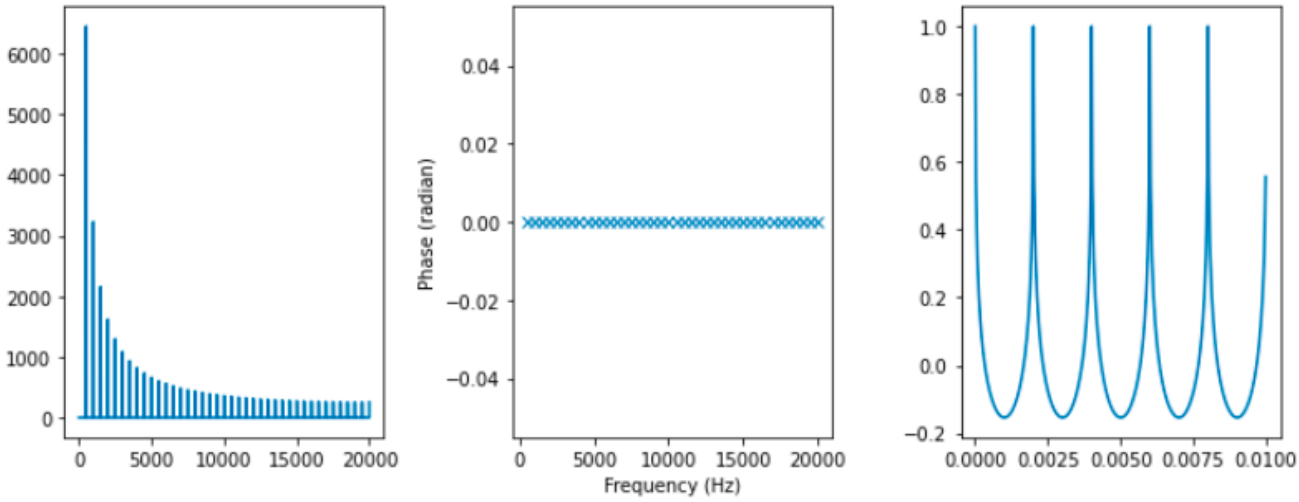

Figure 6: Triangle signal phase = 0

If we rotate phases by some value, we can see that plots changes a lot too, but again we cannot hear any difference with the original signal.

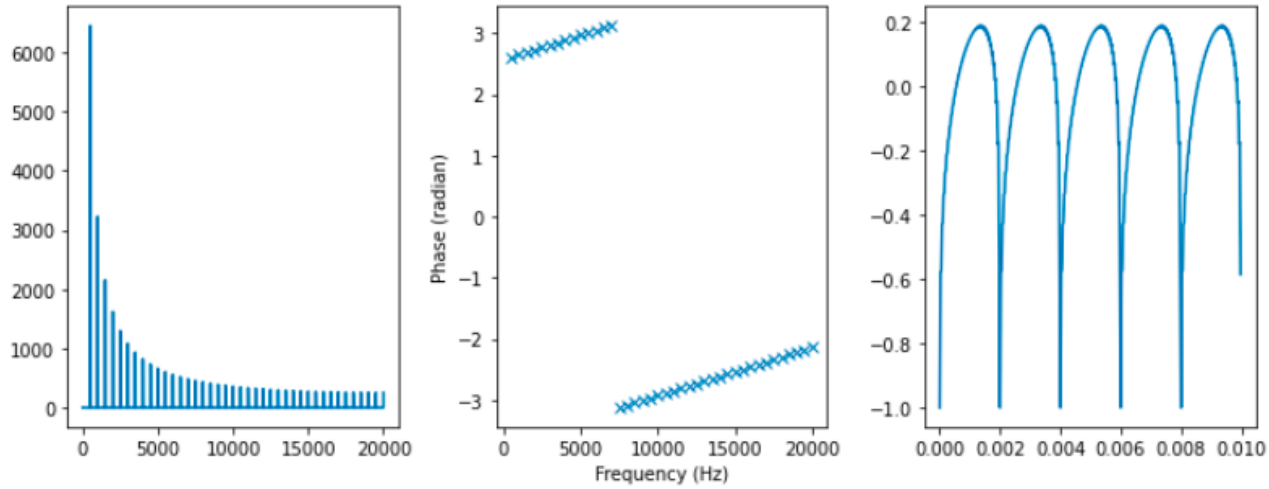

Figure 7: Triangle signal phase rotated

Lastly, let's set every angle to the random value. We can see, that signal doesn't looks like the triangle one at all, but again we cannot hear any difference.

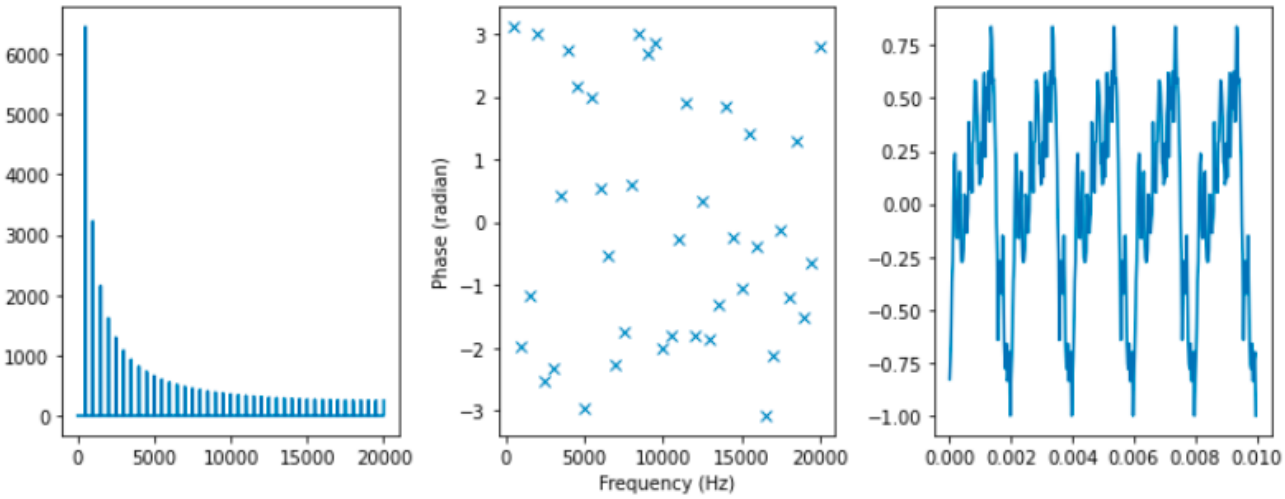

Figure 8: Triangle signal phase = random

But this happens not to the all signals. If we will use more complex signal like oboe record, we see other things happen.

Let's look at the original signal's plots.

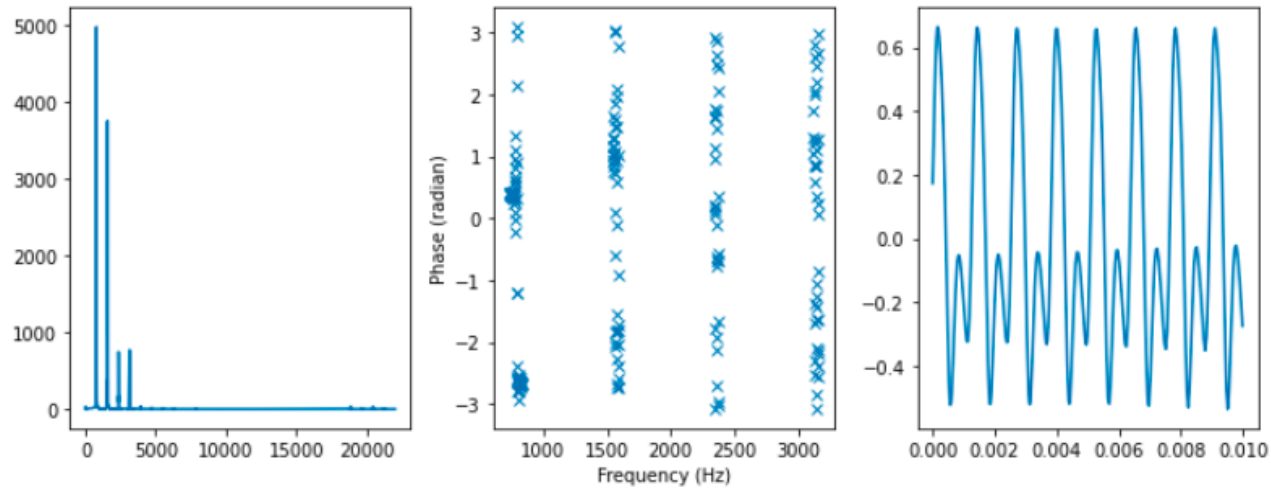

Figure 9: Oboe record plots

By setting every phase to 0 we won't get the same sound, as it was with triangle sound. Sound changes its volume over the time.

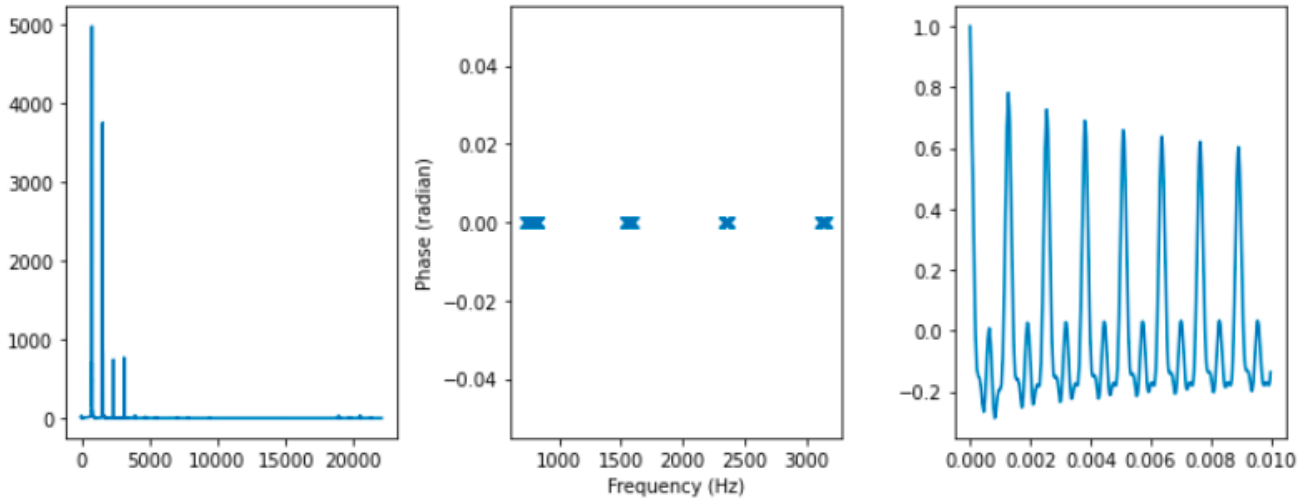

Figure 10: Oboe record phase = 0

By rotating the signal by some angle we won't get this effect, and plot looks like the original one.

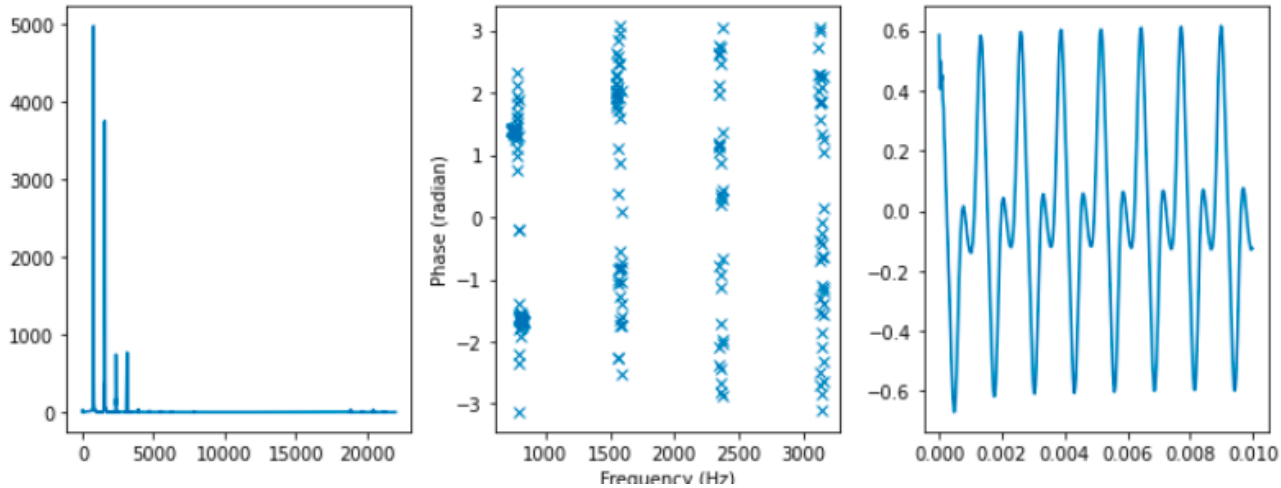

Figure 11: Oboe record phase rotated

However, randomizing the phases returns this effect. Plot also looks ”wrong”.

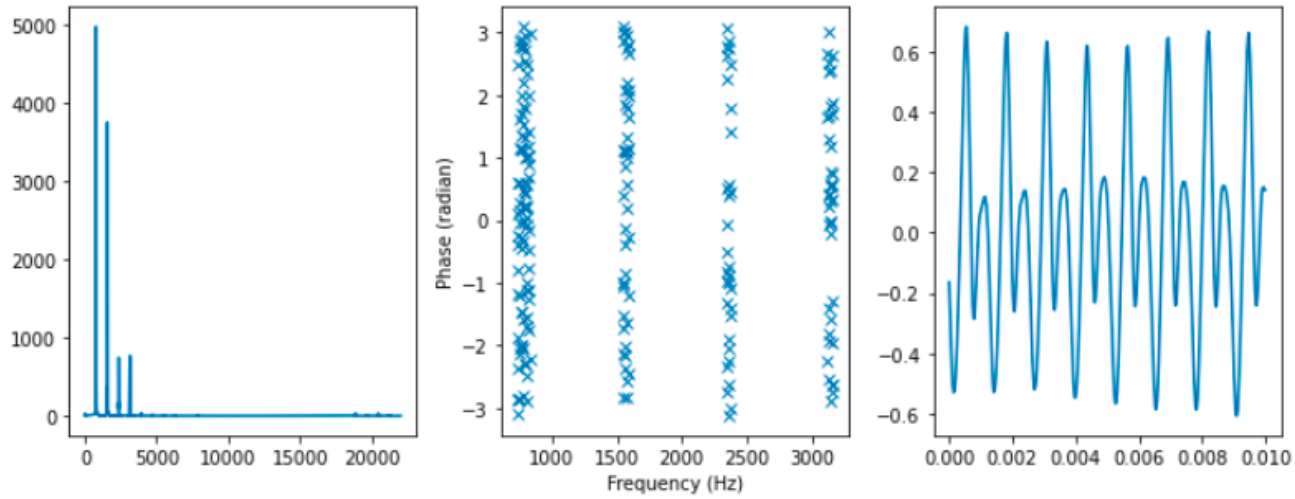

Figure 12: Oboe record phase randomized

Next, we can apply this method to research a saxophone record. Saxophone spectrum looks like triangle signal, but has a low amplitude base frequency.

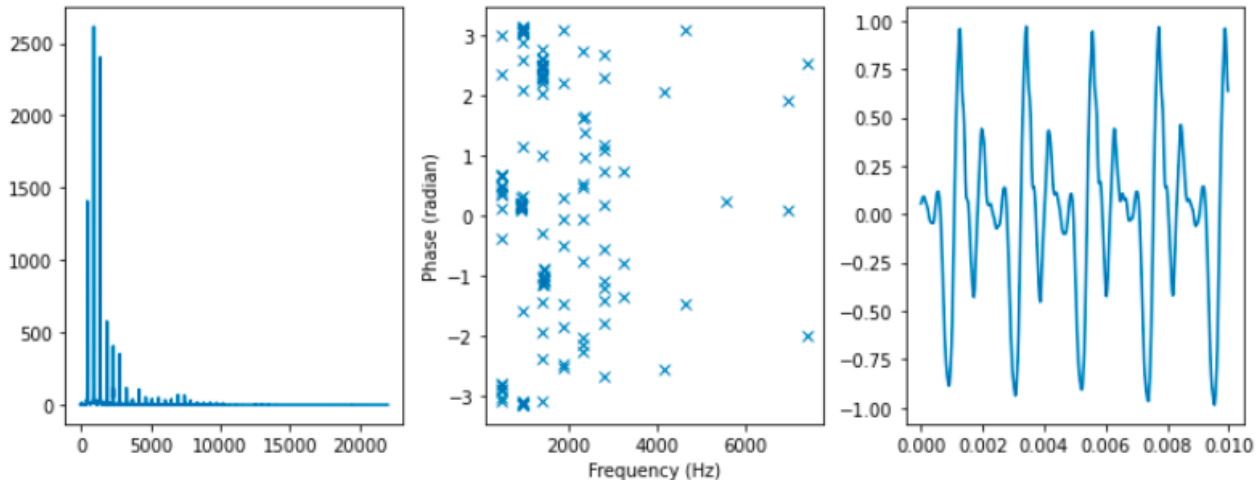

Figure 13: Saxophone record plots

After hearing its audio after each operation we can say, that saxophone's audios have same behavior as oboe's does. Let's remove base frequency from saxophone signal.

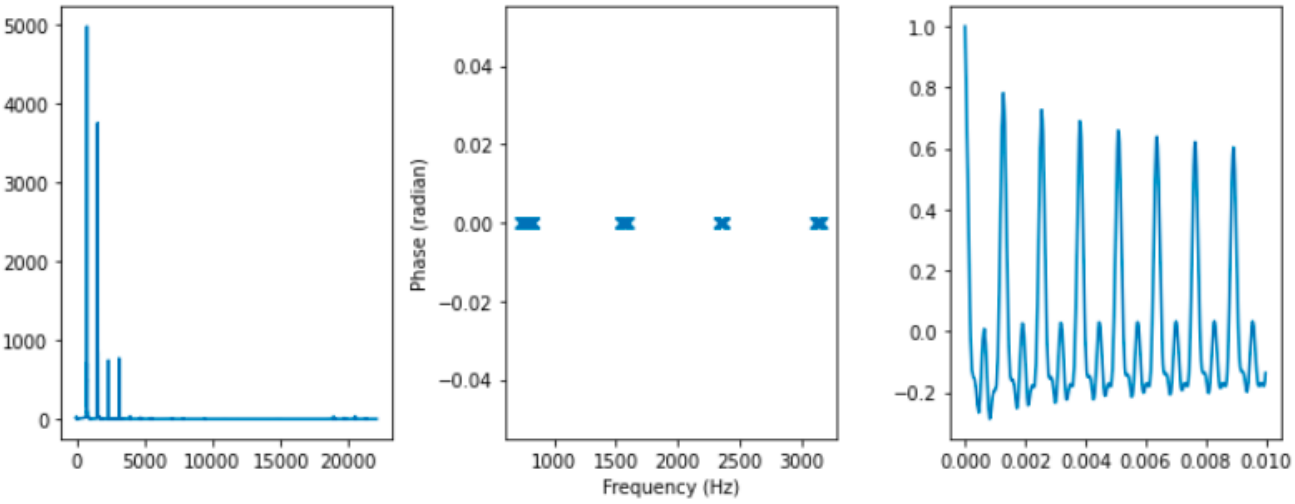

Figure 14: Saxophone record without base frequency

After hearing its audio we can say, that now it behaves like triangle signal does. After all, we can say, that simple signals doesn't changes a lot if their phases are being changed in some way. However, more complex signals changes after their phases was changed.

4 Conclusion

We've learned, what is DCT and how we can use it by compressing the signal. Also we've learned different ways of estimating an amplitude of a signal by knowing it's frequencies and benchmarked them. Last, we've learned how phase affects the signal's perception by a human ear.