Санкт-Петербургский государственный политехнический
университет Петра Великого

**Высшая школа интеллектуальных систем и
суперкомпьютерных технологий**

Лабораторная работа

Дискретное косинусное преобразование

Выполнил студент гр. 3530901/80201 И.С. Иванов

Преподаватель: Н.В. Богач

Санкт-Петербург
2021

Содержание

Список иллюстраций

Листинги

1 Упражнение №1

В первом упражнении необходимо проверить, что `analyze1` требует времени пропорционально n^3, а `analyze2` пропорционально n^2 путем запуска их с несколькими разными массивами. Для этого необходимо воспользоваться `timeint`.

Необходимо создать сигнал на основе некоррелируемого гауссовского шума

```
from thinkdsp import UncorrelatedGaussianNoise

signal = UncorrelatedGaussianNoise()
noise = signal.make_wave(duration=1.0, framerate=16384)
noise.ys.shape

```

Листинг 1: Создание сигнал на основе некоррелируемого гауссовского шума

Создадим функция, которая будет брать и отображать массив результатов, а так же выстраивать их в прямую линию

```
from scipy.stats import linregress

loglog = dict(xscale='log', yscale='log')

def plot_bests(ns, bests):
    plt.plot(ns, bests)
    decorate(**loglog)

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

    return slope

```

Листинг 2: Функция plot_bests

Создадим функция `analyze1`

```
PI2 = np.pi * 2

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

```

Листинг 3: Функция analyze1

Создадим тестовые данные.

```
ns = 2 ** np.arange(6, 13)

```

Листинг 4: Создание тестовых данных

Далее используем данную функцию.

```
results = []
for N in ns:
    print(N)
    ts = (0.5 + np.arange(N)) / N
    freqs = (0.5 + np.arange(N)) / 2
    ys = noise.ys[:N]
    result = %timeit -r1 -o func(ys, freqs, ts)
    results.append(result)

bests = [result.best for result in results]

plot_bests(ns, bests)

```

Листинг 5: Тестирование функции

```
64
94 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 10000 loops each)
128
260 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 1000 loops each)
256
2.18 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
512
8.18 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
1024
36.6 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 10 loops each)
2048
183 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 10 loops each)
4096
770 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

2.2137508528116374

```

Листинг 6: Получившиеся данные

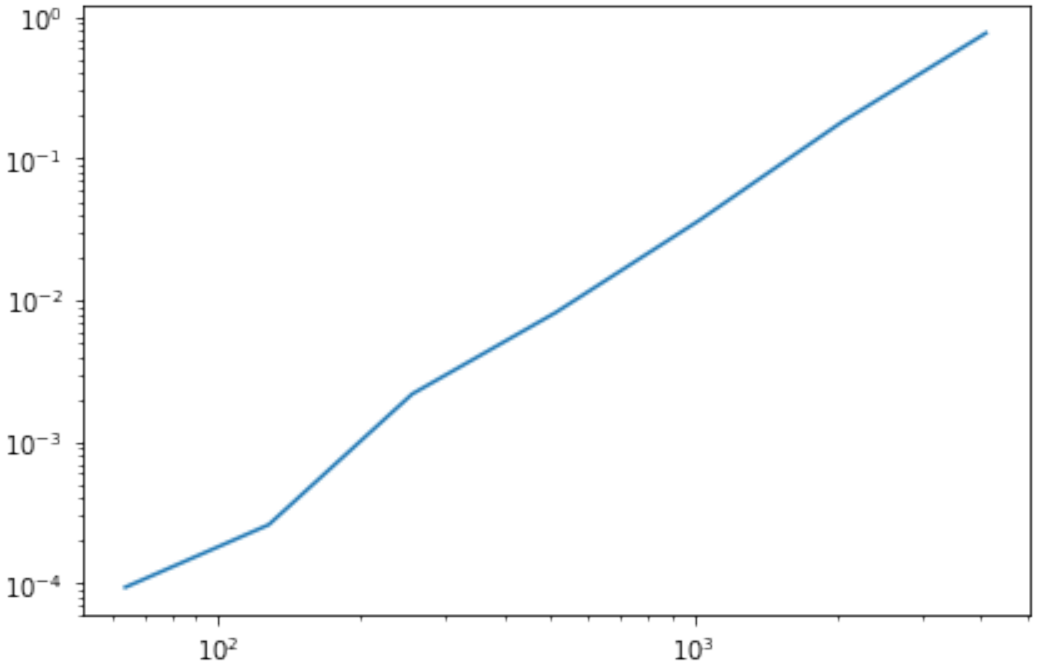

Рис. 1: Получившийся график

Создадим функцию `analyze2`.

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

```

Листинг 7: Функция analyze2

```
64
45 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 10000 loops each)
128
204 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 1000 loops each)
256
1.35 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1000 loops each)
512
4.19 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
1024
18.9 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
2048
78.1 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 10 loops each)
4096
261 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

2.088196535313128

```

Листинг 8: Получившиеся данные

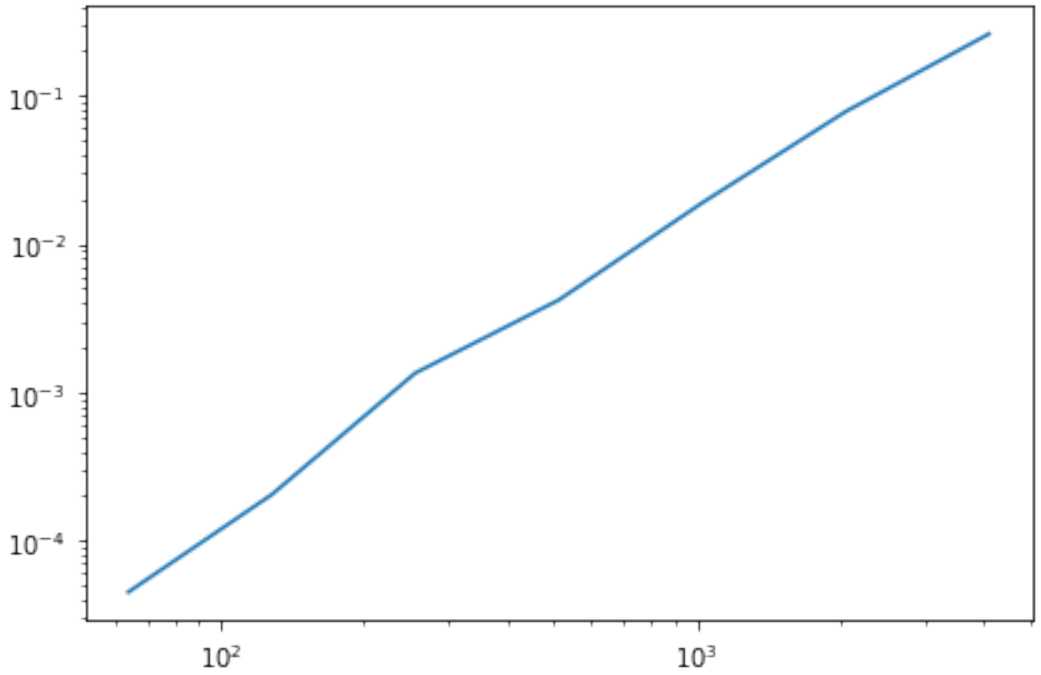

Рис. 2: Получившийся график

Отобразим два графика на одном для сравнения.

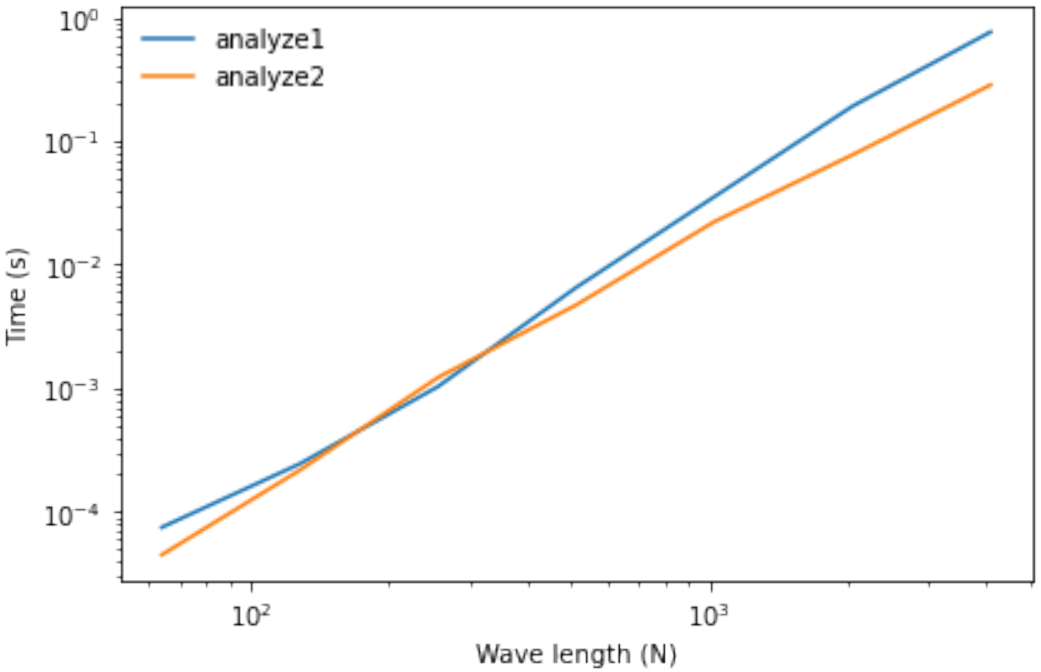

Рис. 3: Совмещенный график

2 Упражнение №2

Во втором упражнении необходимо реализовать версию алгоритма ”DCT”для сжатия звука

Скачаем звук с сайта freesound.org. Прочитаем его. Выделим сегмент длительностью 0.5 секунд.

Далее выведем график ”DCT”

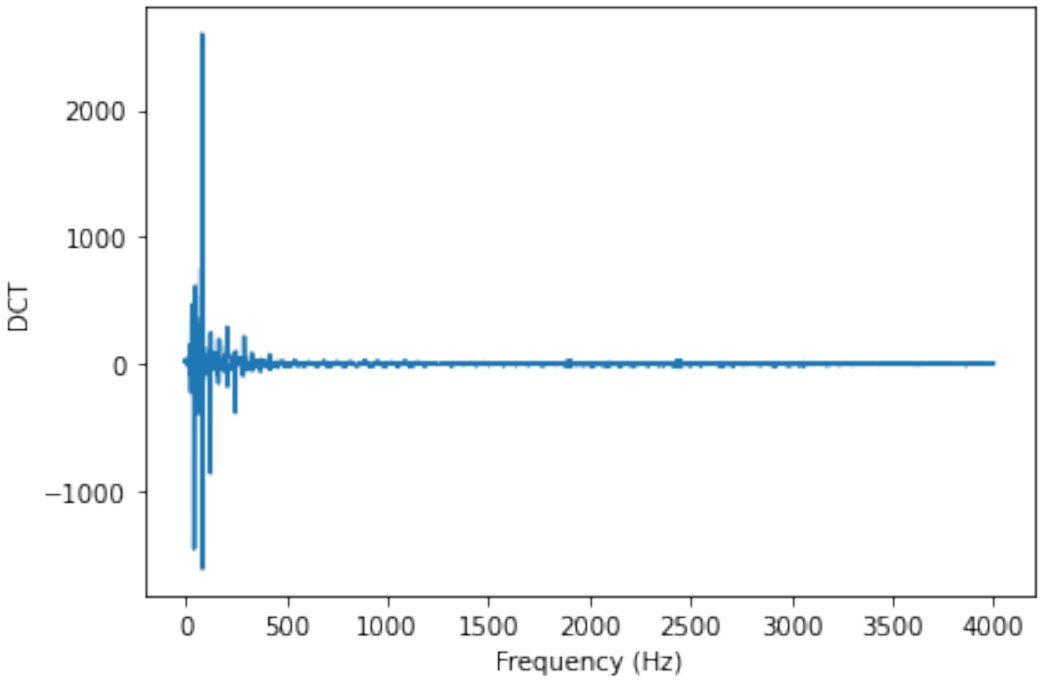

Рис. 4: График DCT сегмента

На графике видно, что в сегменте много точек с нулевой амплитудой. Напишем функцию `compress` для обнуления элементов ниже порога `thresh`.

```
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)
    print(count, n, 100 * count / n, sep='\t')

```

Листинг 9: Функция compress

Применим функцию к выделенному сегменту.

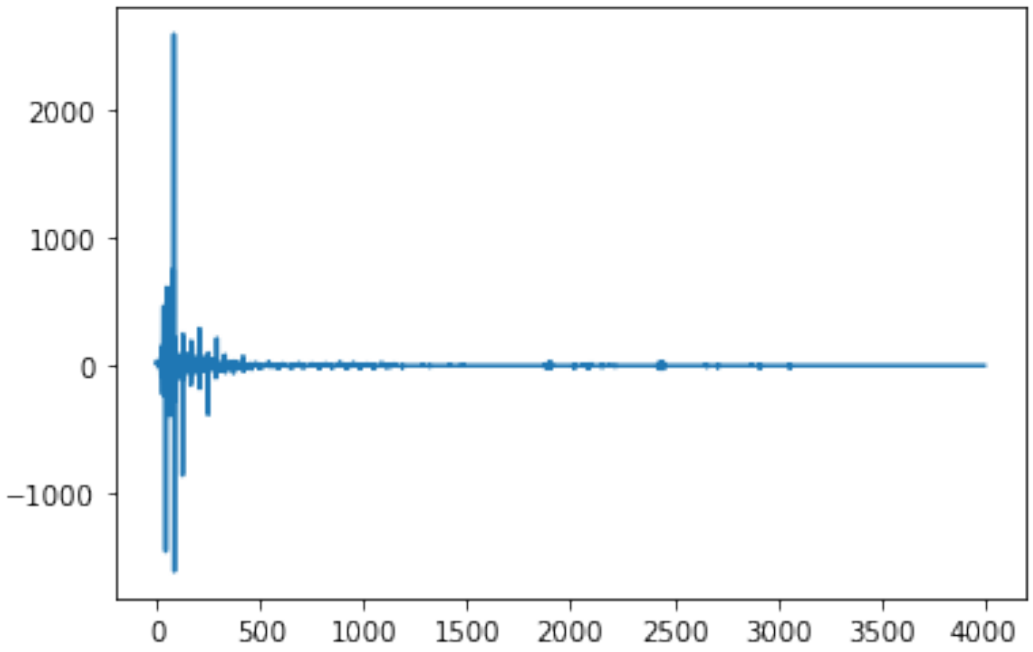

Рис. 5: График сегмента после DCT

Графики почти не отличаются. При прослушивании результата, изменения почти не различимы.

Для сжатия более длинного фрагмента необходимо написать функцию `make_dct_sp`

```
from thinkdsp import Spectrogram

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)

```

Листинг 10: Функция make_dct_spectrogram

Теперь необходимо создать спектрограмму ”DCT”с помощью только что созданной функции и применить `compress` для каждого сегмента.

```
spectro = make_dct_spectrogram(wave, seg_length=1024)
for t, dct in sorted(spectro.spec_map.items()):
    compress(dct, thresh=0.2)
```

Листинг 11: Применение ”DCT”для всего файла

```
999 1024    97.55859375
1004    1024    98.046875
1014    1024    99.0234375
1011    1024    98.73046875
1010    1024    98.6328125
1012    1024    98.828125
1012    1024    98.828125
778 1024    75.9765625
540 1024    52.734375
527 1024    51.46484375
555 1024    54.19921875
568 1024    55.46875
588 1024    57.421875
647 1024    63.18359375
689 1024    67.28515625
708 1024    69.140625
767 1024    74.90234375
824 1024    80.46875
919 1024    89.74609375
951 1024    92.87109375
```

Листинг 12: Пример вывода функции label

Прослушав получившийся сигнал и исходный, никаких изменений не слышно. Управление параметром `thresh` влияет на получаемый после сжатия шум.

3 Упражнение №3

В третьем упражнении нам необходимо запустить файл `phase.ipynb`, пройтись по всем примерам, после выбрать другой сегмент исходного аудио файла и провести с ним те же манипуляции.

Для проведения действий из файла необходимо взять функции `plot_angle` и `plot_three`.

```
def plot_angle(spectrum, thresh=1):
    angles = spectrum.angles
    angles[spectrum.amps < thresh] = np.nan
    plt.plot(spectrum.fs, angles, 'x')
    decorate(xlabel='Frequency (Hz)',
             ylabel='Phase (radian)')

```

Листинг 13: Функция plot_angle

```
def plot_three(spectrum, thresh=1):
    plt.figure(figsize=(10, 4))
    plt.subplot(1,3,1)
    spectrum.plot()
    plt.subplot(1,3,2)
    plot_angle(spectrum, thresh=thresh)
    plt.subplot(1,3,3)
    wave = spectrum.make_wave()
    wave.unbias()
    wave.normalize()
    wave.segment(duration=0.01).plot()
    wave.make_audio()

```

Листинг 14: Функция plot_three

Возьмем фрагмент длительностью 1 секундой, начиная с 5 секунды. Вычислим его спектр и вызовем функцию `plot_three`.

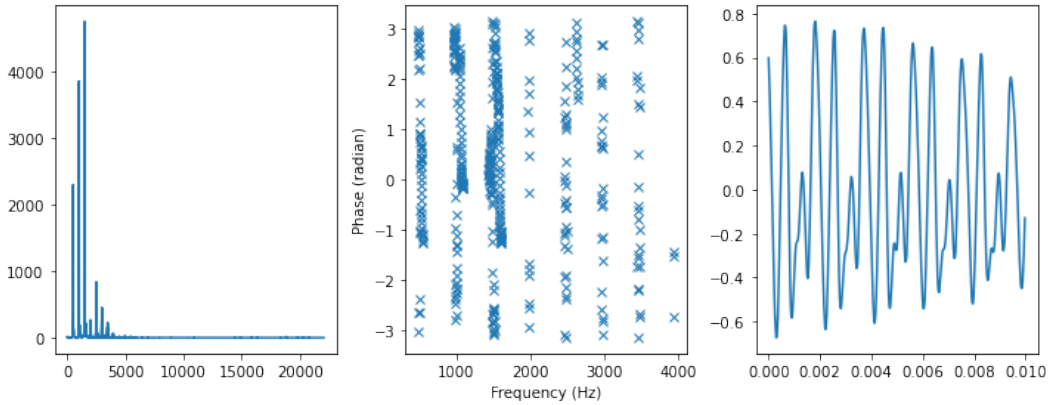

Рис. 6: График спектра

Добавим функцию `zero_angle`, выдающая результат, в котором `angle` = 0.

```
def zero_angle(spectrum):
    res = spectrum.copy()
    res.hs = res.amps
    return res

```

Листинг 15: Функция zero_angle

Получим спектр с помощью `zero_angle` и выведем графики с помощью `plot_three`.

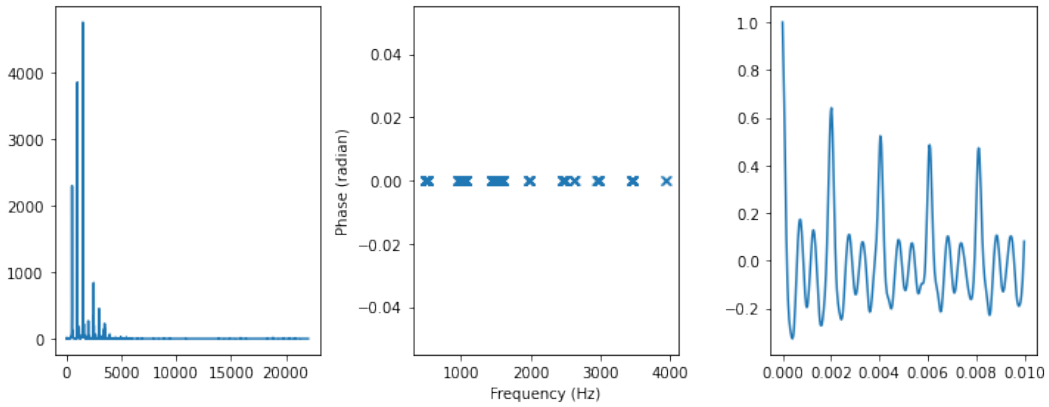

Рис. 7: Графики полученные с помощью zero_angle и plot_three

Добавим функцию `rotate_angle`, выдающая результат, в котором `angle` изменен на 1 радиан.

```
def rotate_angle(spectrum, offset):
    res = spectrum.copy()
    res.hs *= np.exp(1j * offset)
    return res

```

Листинг 16: Функция rotate_angle

Получим спектр с помощью `rotate_angle` и выведем графики с помощью `plot_three`.

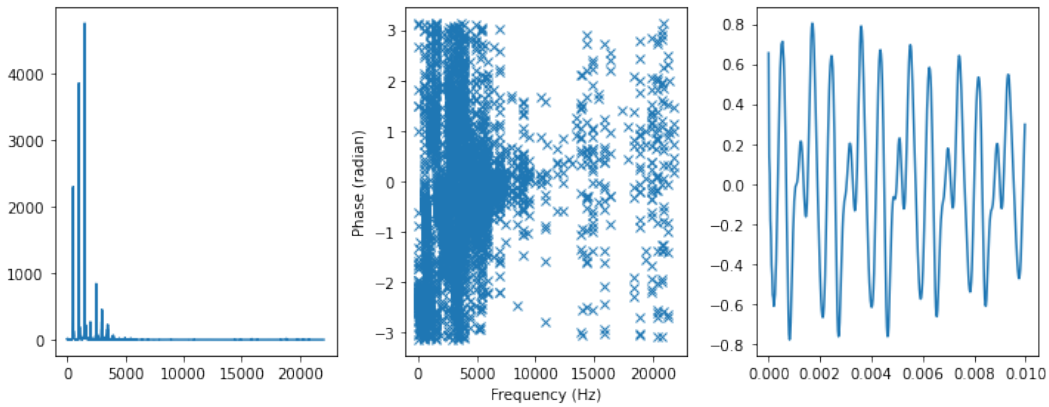

Рис. 8: Графики полученные с помощью rotate_angle и plot_three

Добавим функцию `random_angle`, выдающая результат, в котором `angle` имеет случайное значение.

```
PI2 = np.pi * 2

def random_angle(spectrum):
    res = spectrum.copy()
    angles = np.random.uniform(0, PI2, len(spectrum))
    res.hs *= np.exp(1j * angles)
    return res

```

Листинг 17: Функция random_angle

Получим спектр с помощью `random_angle` и выведем графики с помощью `plot_three`.

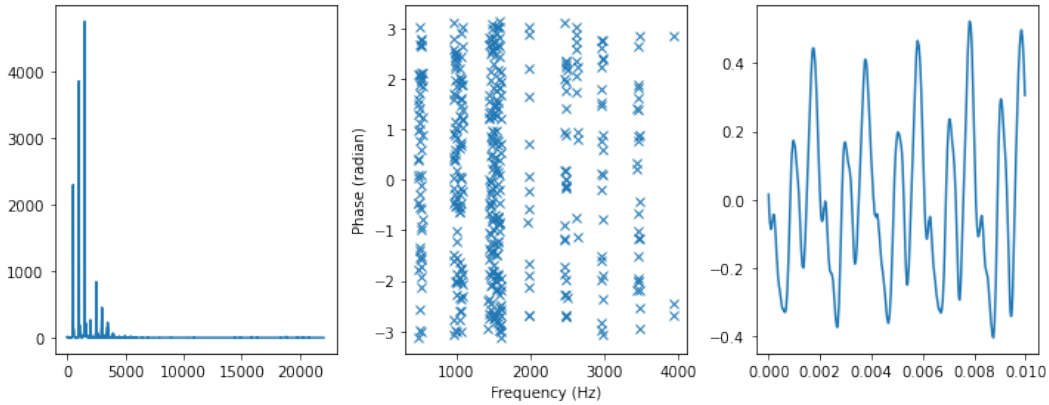

Рис. 9: Графики полученные с помощью random_angle и plot_three

Из полученных результатов можно сделать вывод, что при рандомизация немного приглушила звук, а установка `angle` в ноль искажает звук.

4 Выводы

В результате выполнения данной лабораторной работы мы изучили, понятие DCT. Были проверены функции `analyze1` и `analyze2`. Была создана функция `make_dct_spectrogram` для сжатия звуковой дорожки. Было изучено понятие `angle` и закреплено его влияние на практике.