Санкт-Петербургский государственный политехнический
университет Петра Великого

**Высшая школа интеллектуальных систем и
суперкомпьютерных технологий**

Отчёт по лабораторной работе №6
Дисциплина: Телекоммуникационные технологии
Тема: Дискретное косинусное преобразование

Выполнил студент гр. 3530901/80201 В.А. Пучкина

Преподаватель: Н.В. Богач

Санкт-Петербург
2021

Содержание

Список иллюстраций

Листинги

1 Упражнение 6.1

В этом упражнении необходимо провести следующую проверку: запускать функции `analyze` и `analyze2` с разными массивами и засекать время работы, после чего вывести зависимость времени работы от размера массива на логарифмической шкале. То же самое необходимо проделать для `scipy.fftpack.dct`.

Сначала создадим шумовой сигнал.

```
signal = UncorrelatedGaussianNoise()
noise = signal.make_wave(duration=1, framerate=16384)
```

Листинг 1: Создание шумового сигнала.

Теперь определим необходимые функции:

- `plot_bests` для отображения массива результата,
- `run_speed_test` для запуска теста,
- `analyze1`,
- `analyze2`.

```
def plot_bests(ns, bests):
    plt.plot(ns, bests)
    decorate(xscale='log', yscale='log')

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

Листинг 2: Функция `plot_bests`.

```
def run_speed_test(ns, func):
    results = []
    for N in ns:
        print(N)
        ts = (0.5 + numpy.arange(N)) / N
        freqs = (0.5 + numpy.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]
    return bests
```

Листинг 3: Функция `run_speed_test`.

```
PI2 = numpy.pi * 2

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

Листинг 4: Функция `analyze1`.

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

Листинг 5: Функция `analyze2`.

Итак, подготовим данные для тестирования. Для чистоты эксперимента будем использовать одинаковые данные для всех тестируемых функций.

```
ns = 2 ** numpy.arange(7, 14)
```

Листинг 6: Формирование тестовых данных.

Теперь протестируем `analyze1`.

```
bests = run_speed_test(ns, analyze1)
plot_bests(ns, bests)
```

Листинг 7: Тестирование `analyze1`.

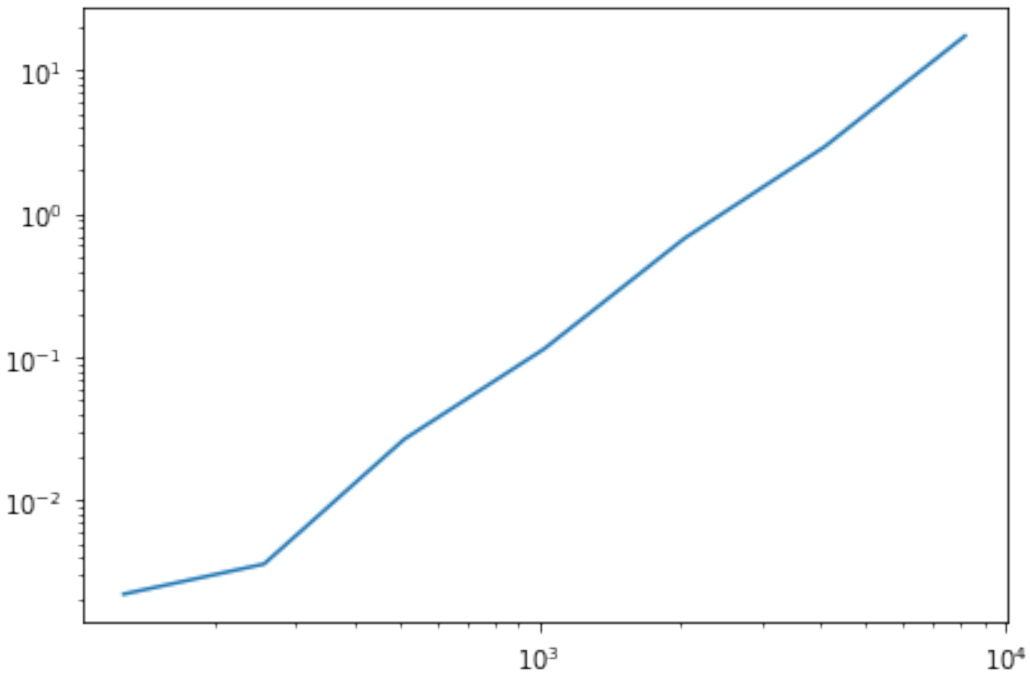

Рис. 1: Зависимость времени работы от размера (`analyze1`).

Рассчитанный наклон составил примерно 2.2, что не соответствует ожиданиям. Такой результат мог возникнуть из-за выборки в массиве.

Протестируем `analyze2`.

```
bests2 = run_speed_test(ns, analyze2)
plot_bests(ns, bests2)
```

Листинг 8: Тестирование `analyze1`.

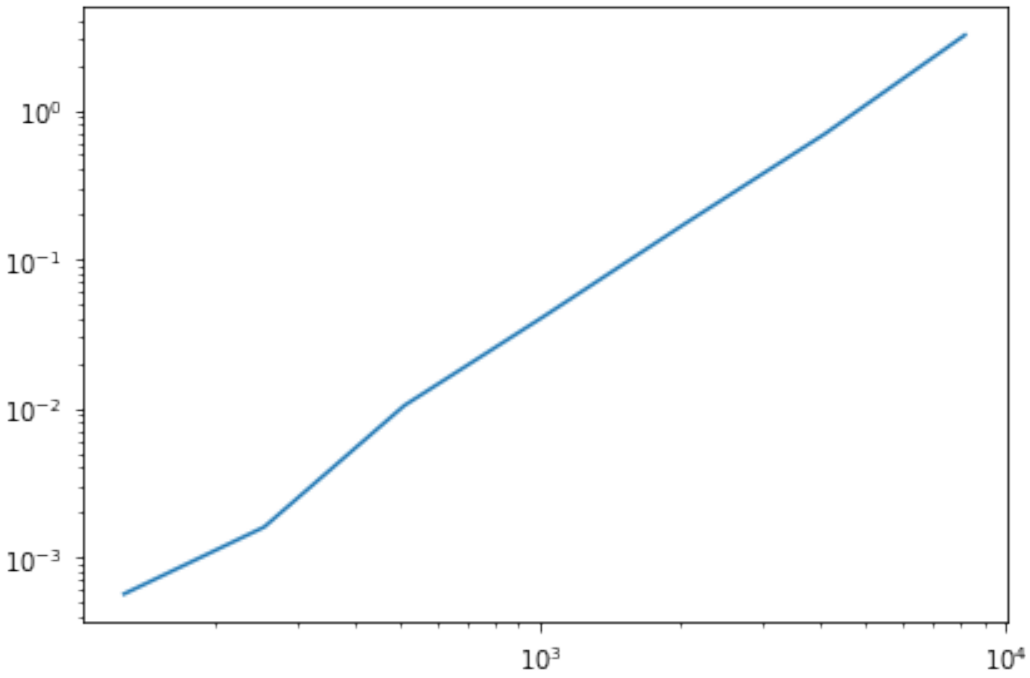

Рис. 2: Зависимость времени работы от размера (`analyze2`).

Для `analyze2` рассчитанный наклон составил примерно 2.1, что совпало с ожиданиями.

И наконец, протестируем `scipy_dct`.

```
bests2 = run_speed_test(ns, analyze2)
plot_bests(ns, bests2)
```

Листинг 9: Функция `scipy_dct`.

Эта реализация работает ещё быстрее. По зависимости времени работы от размера (Рис.3) видно, что зависимости нелинейна (график изогнут). Это означает, что либо мы еще не видели асимптотическое поведение, либо асимптотическое поведение не является простым показателем n. Её время выполнения пропорционально $nlogn$.

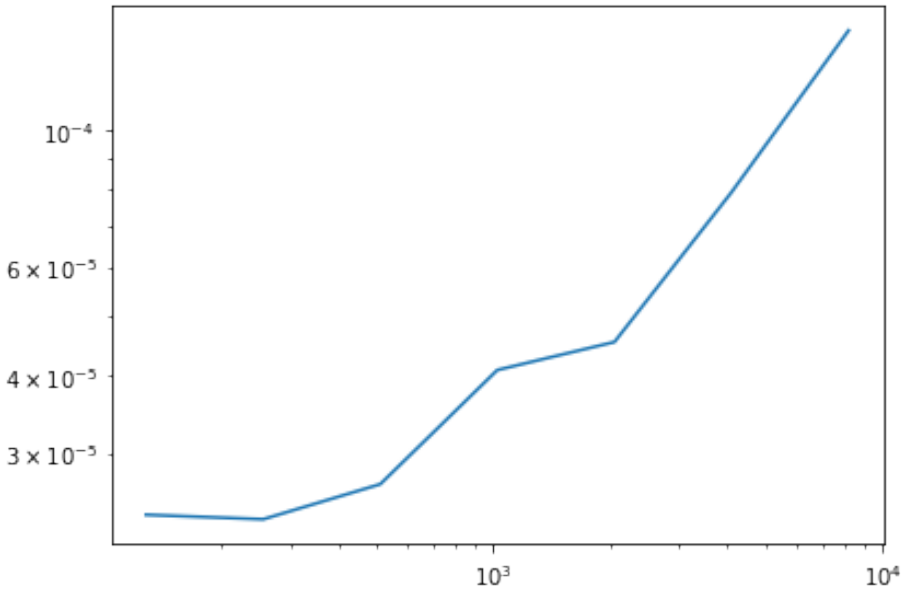

Рис. 3: Зависимость времени работы от размера (`scipy_dct`).

Теперь сравним все полученные результаты, для этого выведем их не один график.

```
plt.plot(ns, bests, label='analyze1')
plt.plot(ns, bests2, label='analyze2')
plt.plot(ns, bests3, label='fftpack.dct')
decorate(xlabel='Wave length (N)', ylabel='Time (s)', xscale='log', yscale='log')
```

Листинг 10: Сравнение полученных результатов.

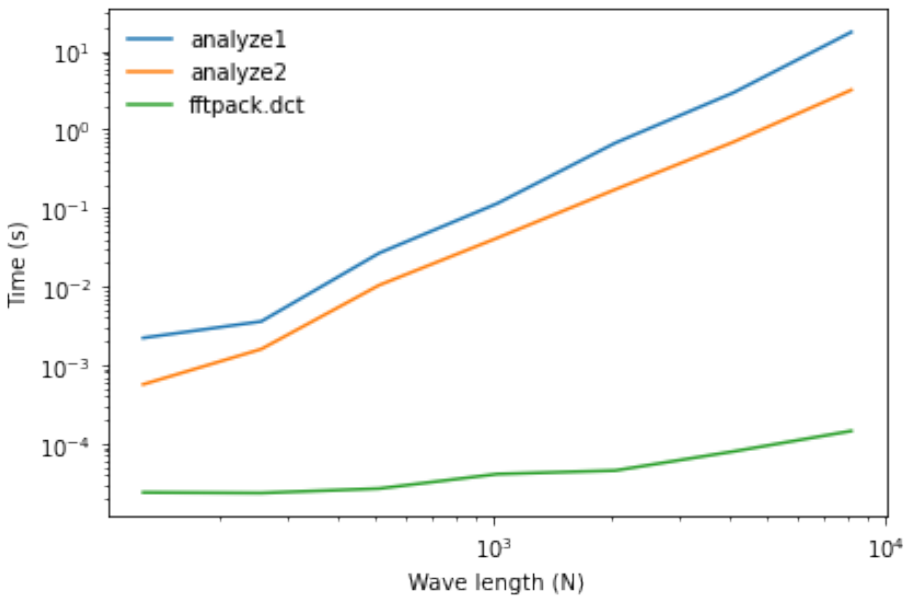

Рис. 4: Зависимости времени работы от размера массива для разных функций.

Таким образом, из полученного графика (Рис.4) видно, что `scipy_dct` является самой быстрой реализацией.

В ходе выполнения данного упражнения были получены зависимости времени работы от размера массива для функций `analyze1`, `analyze2` и `scipy_dct`. Также был получен общий график зависимостей. По полученным результатам был сделан вывод, что `scipy_dct` является самой быстрой реализацией.

2 Упражнение 6.2

В этом упражнении необходимо реализовать алгоритм для сжатия с помощью ДКП. После чего применить полученный алгоритм на записи музыки или речи.

Итак, для проверки будем использовать запись гитары, скаченную отсюда.

```
guitar_wave = read_wave('resources/Sounds/task2_serylis_guitar_chord.wav')
guitar_wave.make_audio()
```

Листинг 11: Чтение файла и получение `wave`.

```
guitar_segment = guitar_wave.segment(start=0.7, duration=0.5)
guitar_segment.normalize()
guitar_segment.make_audio()
```

Листинг 12: Выбор сегмента.

Теперь получим ДКП сегмента.

```
guitar_dct = guitar_segment.make_dct()
guitar_dct.plot(high=1500)
decorate(xlabel='Frequency (Hz)', ylabel='DCT')
```

Листинг 13: Получение ДКП.

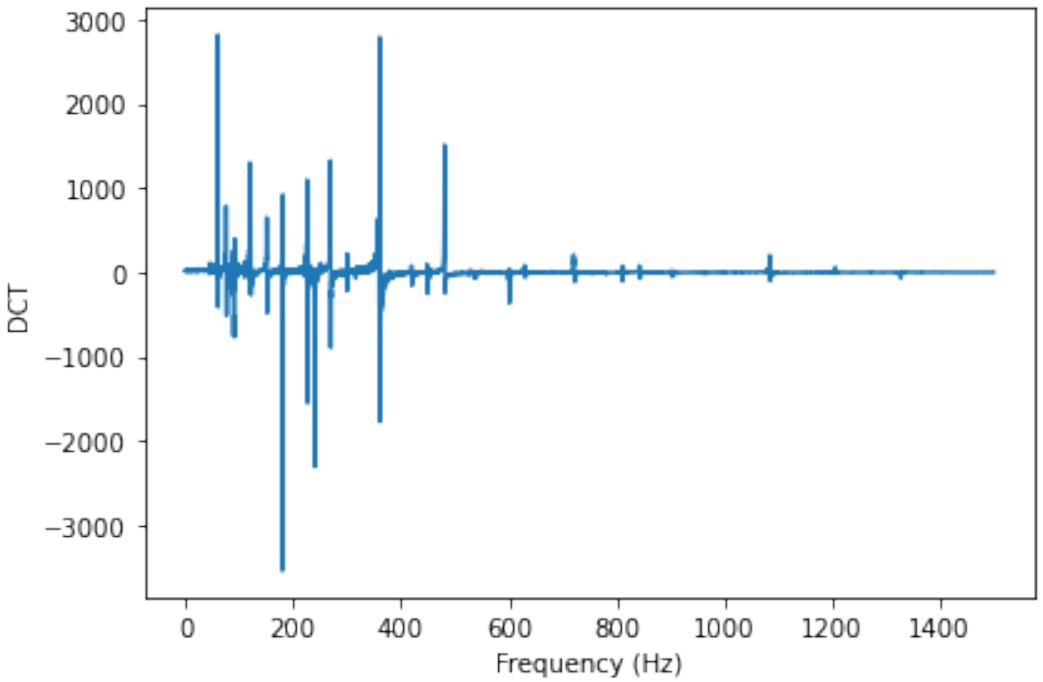

Рис. 5: ДКП сегмента.

Теперь реализуем алгоритм сжатия с помощью ДКП.

```
def compress(dct, thresh=1):
    count = 0
```

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

    n = len(dct.amps)
    print(count, n, 100 * count / n, sep='\t')
```

Листинг 14: Функция `compress` для сжатия.

Применим функцию `compress` на нашем сегменте.

```
compress(guitar_dct, thresh=10)
guitar_dct.plot(high=1500)
guitar_segment2 = guitar_dct.make_wave()
guitar_segment2.make_audio()
```

Листинг 15: Сжатие сегмента.

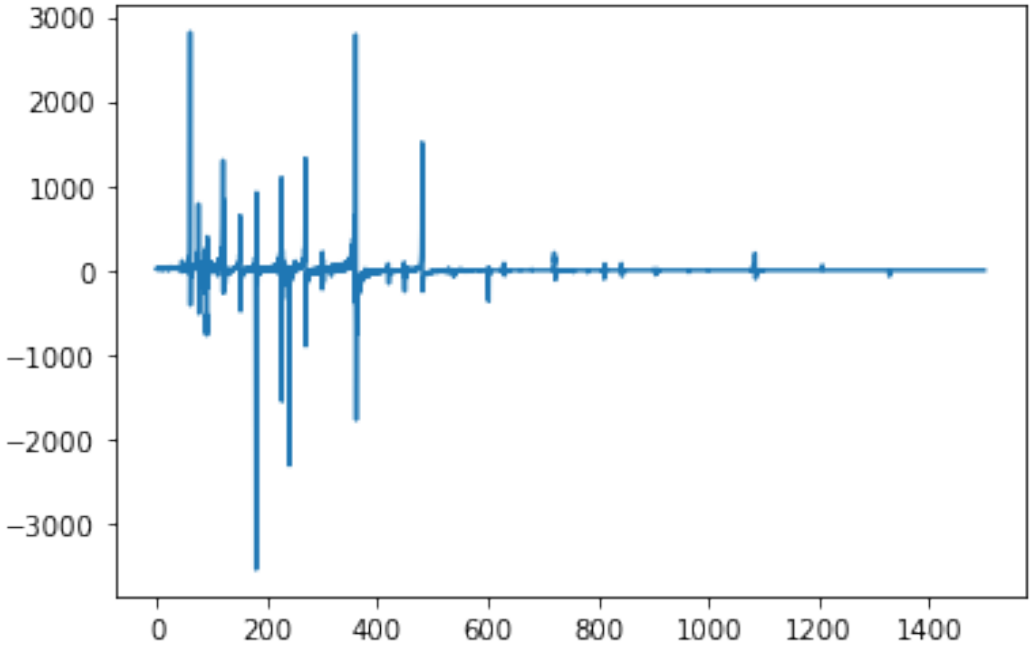

Рис. 6: ДКП сжатого сегмента.

При сравнении оригинального сегмента с полученным разница не слышима, несмотря на то что было удалено около 95% гармоник.

Теперь реализуем функцию для сжатия большого сегмента. Она аналогична `wave.make_spectrogram`, только в ней используется ДКП.

```
def make_dct_spectrogram(wave, seg_length):
    window = numpy.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 16: Функция `make_dct_spectrogram` для сжатия `wave`.

Используем полученную функцию на `wave` всей записи.

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

Листинг 17: Сжатие `wave` всей записи.

Прослушаем результат.

```
guitar_wave2 = guitar_spectrogram.make_wave()
guitar_wave2.make_audio()
```

Листинг 18: Получение аудио результата.

При сравнении полученного аудио с оригинальным разница не слышна, хотя на разных сегментах сжатие составляло от 50 до 99%.

В ходе выполнения данного упражнения был реализован алгоритм для сжатия с помощью ДКП. Затем полученный алгоритм был использован на сегменте записи аккорда гитары. Несмотря на то что компрессия составила около 95%, звучание не изменилось. Затем был использован алгоритм для сжатия `wave`. Компрессия на разных сегментах составляла от 50 до 99%, но звук по прежнему воспринимается так же.

3 Упражнение 6.3

В данном упражнении необходимо изучить файл `phase.ipnyb`, в котором исследуется влияние фазы на восприятие звука. После изучения примеров следует выбрать другой сегмент звука и повторить эксперименты.

Итак, сначала определим необходимые функции, которые используются в файле `phase.ipnyb`.

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

Листинг 19: 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()
    display(wave.make_audio())
```

Листинг 20: Функция `plot_three`.

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

Листинг 21: Функция `zero_angle`.

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

Листинг 22: Функция `rotate_angle`.

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

Листинг 23: Функция `random_angle`.

Теперь считаем файл и выберем сегмент, отличный от использующегося в примере.

```
oboe_wave = read_wave('resources/Sounds/task3_thirsk_120_oboe.wav')
oboe_wave.make_audio()
```

Листинг 24: Чтение файла.

```
oboe_segment = oboe_wave.segment(start=2.1, duration=1)
oboe_segment.make_audio()

```

Листинг 25: Выбор сегмента.

Используем на этом сегменте `plot_three` для получения спектра, угловой части спектра и `wave`.

```
oboe_spectrum = oboe_segment.make_spectrum()
plot_three(oboe_spectrum, thresh=50)
```

Листинг 26: Применение `plot_three` к оригинальному сегменту.

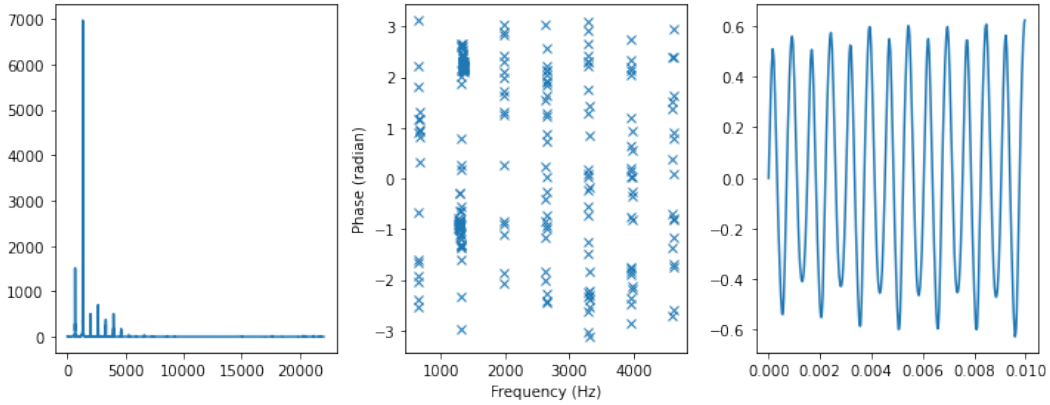

Рис. 7: Спектр, фаза и `wave` оригинального сегмента.

Теперь установим углы на нуль.

```
oboe_spectrum2 = zero_angle(oboe_spectrum)
plot_three(oboe_spectrum2)
```

Листинг 27: Применение `plot_three` к сегменту с нулевыми углами.

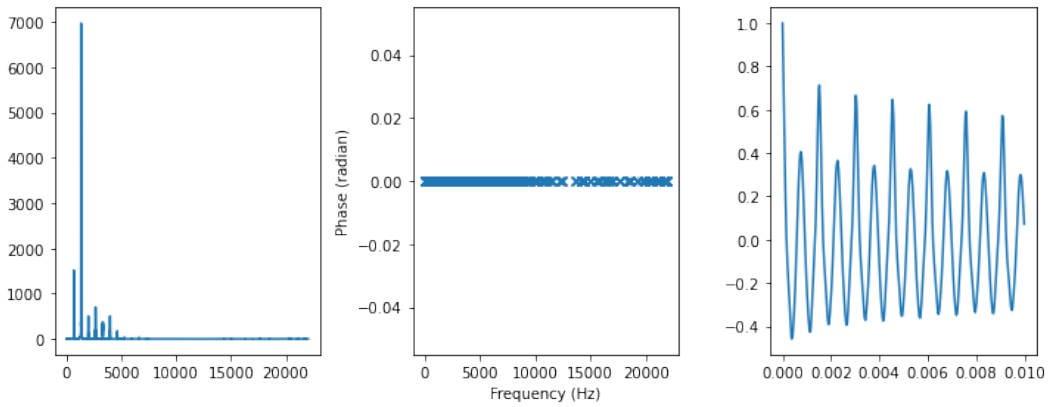

Рис. 8: Спектр, фаза и `wave` сегмента с нулевыми углами.

Можно заметить, что спектр не изменился, а вот форма `wave` изменилась. Изменение фазовой структуры создаёт эффект «звона», когда громкость меняется со временем.

Теперь повернём углы на 1 радиану.

```
oboe_spectrum3 = rotate_angle(oboe_spectrum, 1)
plot_three(oboe_spectrum3)
```

Листинг 28: Применение `plot_three` к сегменту с повёрнутыми углами.

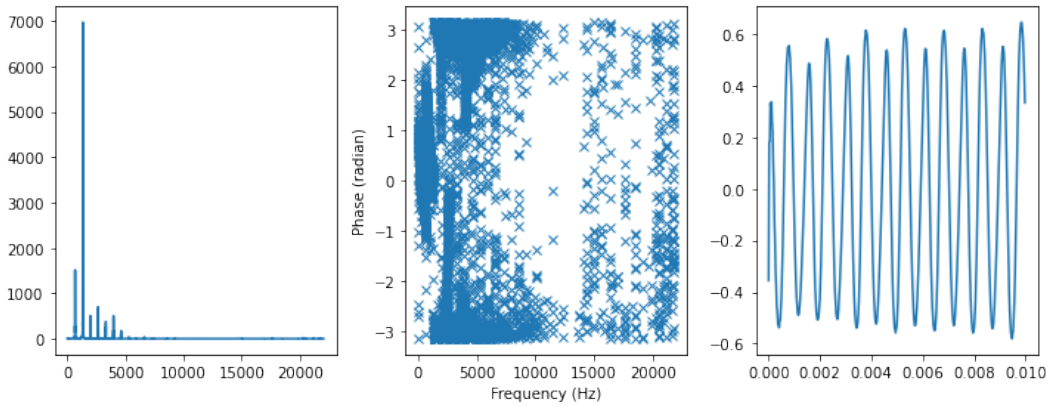

Рис. 9: Спектр, фаза и `wave` сегмента с повёрнутыми углами.

Вращение углов не вызывает эффекта «звона».

Теперь установим ля углов рандомные значения.

```
oboe_spectrum4 = random_angle(oboe_spectrum)
plot_three(oboe_spectrum4)

```

Листинг 29: Применение `plot_three` к сегменту с рандомными углами.

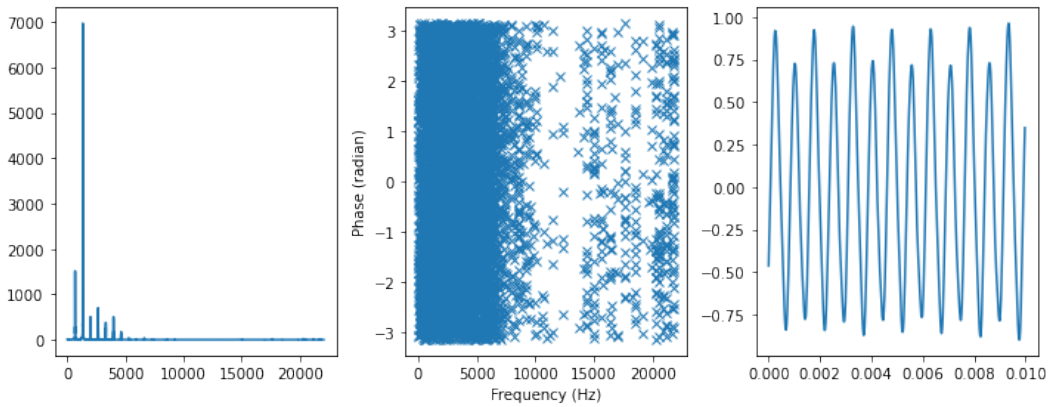

Рис. 10: Спектр, фаза и `wave` сегмента с рандомными углами.

В данном случае рандомные значения углов вызывают небольшой эффект «звона», звук становится более сухим.

В ходе выполнения данного упражнения исследовалось влияние фазы на восприятие звука. Для этого были изучены примеры, представленные в файле `phase.ipnyb`. После чего был выбран другой сегмент и проведён повторный эксперимент. Был сделан вывод, что для звуков с простой гармонической структурой изменения фазовой структуры не заметны на слух. Возможным исключением могут быть звуки с низкой амплитудой на основной частоте.

4 Выводы

В ходе данной лабораторной работы было изучено дискретное косинусное преобразование, были получены навыки по работе с ним. Кроме того, был написан и протестирован алгоритм для сжатия записи. Также было исследовано влияние фазы на восприятие звука с помощью анализа различных звуков и сегментов.