Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

Телекоммуникационные технологии

Отчёт по лабораторной работе №6

Работу выполнил:
Смирнов Л. Д.
Группа:
3530901/80202
Преподаватель:
Богач Н. В.

Санкт-Петербург
2021

Содержание

1. Выполнение работы **3**
1.1. Упражнение 1 3
1.2. Упражнение 2 5
1.3. Упражнение 3 6

2. Выводы **7**

1. Выполнение работы

1.1. Упражнение 1

Для оценки затрачиваемого на работу функций-анализаторов времени я добавил собственно саму тестирующую функцию и функцию для оценки наклона графиков:

```
def run_speed_test(ns, func):
    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]
    return bests

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

    return slope
```

После этого я создал сигнал, на котором будет проводиться тестирование и массив со степенями двойки с 4-ой по 11-ю для использования в тестирующей функциии.

```
signal = UncorrelatedGaussianNoise()
noise = signal.make_wave(duration=1.0, framerate=16384)
ns = 2 ** np.arange(4, 12)
```

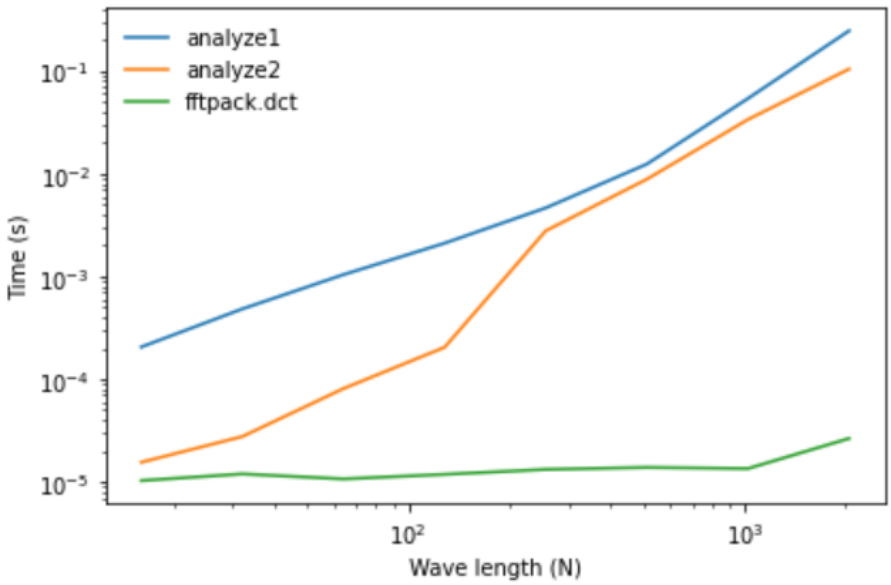
Ниже привдены результаты запуска тестирующей функции для для всех трех функций (analyze1, analyze2 и fftpack.dct):

```
16
206 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 1000 loops each)
32
483 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 1000 loops each)
64
1.04 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1000 loops each)
128
2.11 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
256
4.65 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
512
12.4 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
1024
54 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 10 loops each)
2048
246 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
```

```
16
15.6 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100000 loops each)
32
27.7 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 10000 loops each)
64
81.1 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 10000 loops each)
128
205 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 1000 loops each)
256
2.78 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
512
8.85 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
1024
33.7 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 10 loops each)
2048
105 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 10 loops each)
```

```
16
10.3 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100000 loops each)
32
12 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100000 loops each)
64
10.7 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100000 loops each)
128
11.9 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100000 loops each)
256
13.2 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100000 loops each)
512
13.9 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100000 loops each)
1024
13.5 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 10000 loops each)
2048
26.5 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 10000 loops each)
```

И соответствующие показатели уклона для трех функий:
1.3982672605607114
1.956322093917956
0.13915121272766645
Также для удобства сравнения я вывел все три графика на логарифмическую плоскость для более наглядного сравнения:



Отсюда можно сделать 3 вывода: 1) У функции fftpack.dct лучшие временные затраты. 2) Значение для первой анализирующей функции по неизвестным мне причинам некорректно. 3) Значение для второй функции соответствует ожидаемому.

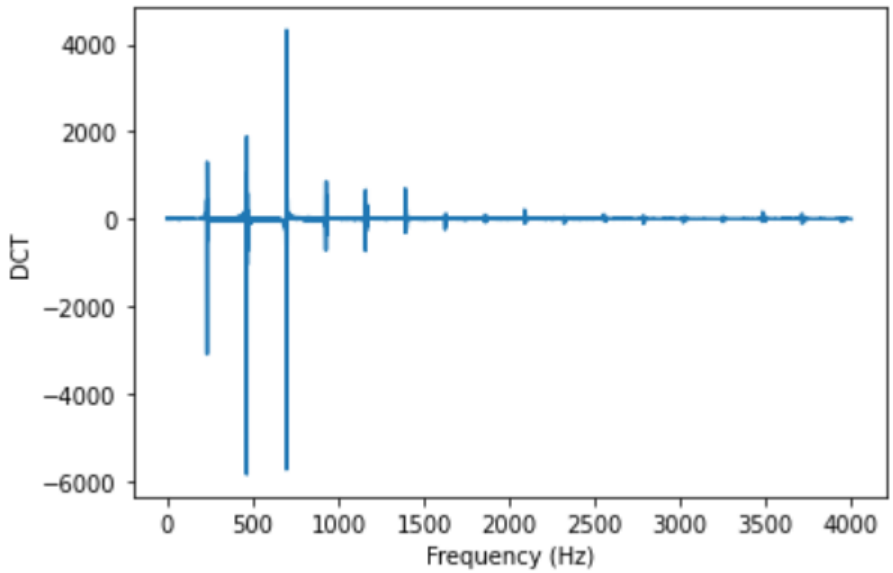
1.2. Упражнение 2

Для этого упражнения я взял фрагмент записи звуков саксофона из предыдущей лабораторной работы.

```
wave = thinkdsp.read_wave('100475__iluppai__saxophone-weep.wav')
segment = wave.segment(start=2.0, duration=0.5)
segment.normalize()
```

Далее я получил ДКП для этого фрагмента и вывел изображение:

```
seg_dct = segment.make_dct()
seg_dct.plot(high=4000)
thinkplot.config(xlabel='Frequency (Hz)', ylabel='DCT')
```



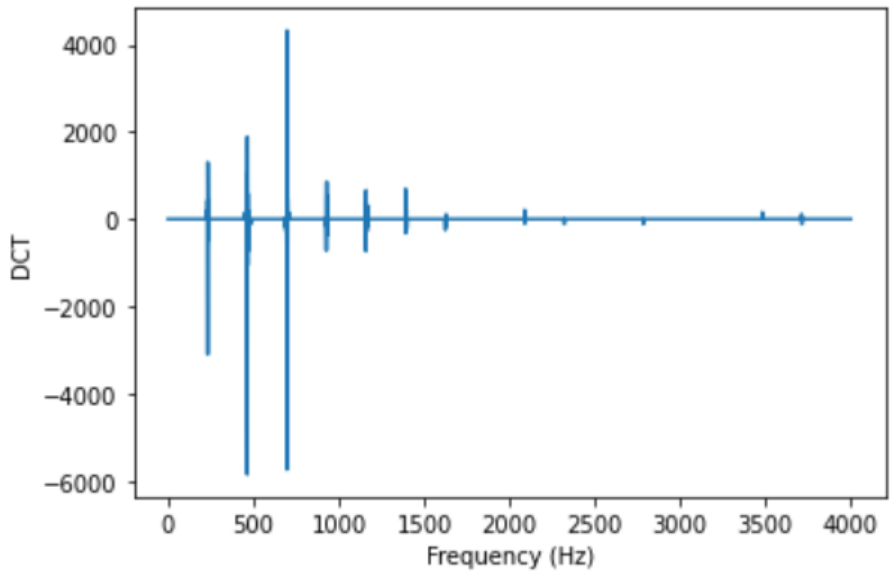
Для обнуления всех частот, которые не играют большой роли в образовании сигнала я добавил следующую функцию:

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

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

Повторное получение ДКП фрагмента и с применением этой функции для отсечения ненужных (в рамках эксперимента) элементов, дает следующую картинку:

Как показало повторное прослушивание, звучание фрагмента при этом практически не изменилось. Для сжатия более длинных фрагментов можно применить функцию, создающую спектрограмму с использованием ДКП:




```
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)
```

Я эту функцию к записи, для компрессии всех фрагментов:

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

Вывод, состоящий из 1024 строк (здесь я его приводить не буду) показал, что большинство фрагментов удалось избавить от 70-90% элементов, причем повторное прослушивание, полученной из спектрограммы волны показало, что на звучании это практически никак не сказалось.

1.3. Упражнение 3

Согласно заданию в книге я прогнал примеры из файла phase.ipynb для разных сегментов звука. Как небольшой вывод/наблюдение могу написать, что удаление компонент звука, которые не играют большой роли, делает структуру сигнала более читаемой, причем зависимость величины фазы компоненты от её частоты практически не разлиима

на слух, будь она линейна или случайна. В то же время фазовая структура звуков, у которых отсутствуют важные частотные компоненты, мы свособны воспринять на слух.

2. Выводы

В ходе выполнения данной лабораторной работы я изучил дискретное косинусное преобразование, прошелся по всем шагам его исследования, а закрепил также его применение на практике.