# Лабораторная работа №6
# Дискретное косинусное преобразование

Смирнов Никита

26 апреля 2021 г.

# Оглавление

# Список иллюстраций

# Листинги

# Глава 1

# Упражнение 6.1

Я добавил тестирующую функцию и функцию для оценки наклона графиков:

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

    bests = [res.best for result in res]
    return bests

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

    return slope
```

Листинг 1.1: Тестирующая функция и функция для оценки наклона графиков

Далее я создал сигнал, на котором будет происходить тестирование и массив со степенями двойки с 4-ой по 11-ю.

```
signal = UncorrelatedGaussianNoise()
```

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

Листинг 1.2: Массив степеней двойки

Ниже я привел резульиаты запуска тестирующей функции для трех функций(analyze1, analyze2, fftpack.dct):

```
16
202 µs ±0 ns per loop (mean ±std. dev. of 1 run, 1000 loops each)
32
463 µs ±0 ns per loop (mean ±std. dev. of 1 run, 1000 loops each)
64
1.32 ms ±0 ns per loop (mean ±std. dev. of 1 run, 1000 loops each)
128
2.62 ms ±0 ns per loop (mean ±std. dev. of 1 run, 100 loops each)
256
6.49 ms ±0 ns per loop (mean ±std. dev. of 1 run, 100 loops each)
512
13.1 ms ±0 ns per loop (mean ±std. dev. of 1 run, 100 loops each)
1024
65.9 ms ±0 ns per loop (mean ±std. dev. of 1 run, 10 loops each)
2048
315 ms ±0 ns per loop (mean ±std. dev. of 1 run, 1 loop each)

16
15.4 µs ±0 ns per loop (mean ±std. dev. of 1 run, 100000 loops
    each)
32
20.8 µs ±0 ns per loop (mean ±std. dev. of 1 run, 10000 loops each)
64
73.9 µs ±0 ns per loop (mean ±std. dev. of 1 run, 10000 loops each)
128
200 µs ±0 ns per loop (mean ±std. dev. of 1 run, 1000 loops each)
256
1.33 ms ±0 ns per loop (mean ±std. dev. of 1 run, 1000 loops each)
512
7.01 ms ±0 ns per loop (mean ±std. dev. of 1 run, 100 loops each)
1024
29.1 ms ±0 ns per loop (mean ±std. dev. of 1 run, 10 loops each)
2048
92 ms ±0 ns per loop (mean ±std. dev. of 1 run, 10 loops each)

16
```

```
8.81 µs ±0 ns per loop (mean ±std. dev. of 1 run, 100000 loops each)
32
9.91 µs ±0 ns per loop (mean ±std. dev. of 1 run, 100000 loops each)
64
10.4 µs ±0 ns per loop (mean ±std. dev. of 1 run, 100000 loops each)
128
11.3 µs ±0 ns per loop (mean ±std. dev. of 1 run, 100000 loops each)
256
9.62 µs ±0 ns per loop (mean ±std. dev. of 1 run, 100000 loops each)
512
13.1 µs ±0 ns per loop (mean ±std. dev. of 1 run, 100000 loops each)
1024
16 µs ±0 ns per loop (mean ±std. dev. of 1 run, 100000 loops each)
2048
35.7 µs ±0 ns per loop (mean ±std. dev. of 1 run, 10000 loops each)
```

Листинг 1.3: Результат работы

Также приведу показатели уклона трех функций: 1.4429081741104806 1.9340330653787678 0.21817871893410304

И для наглядности я вывыел три графика для сравнения:

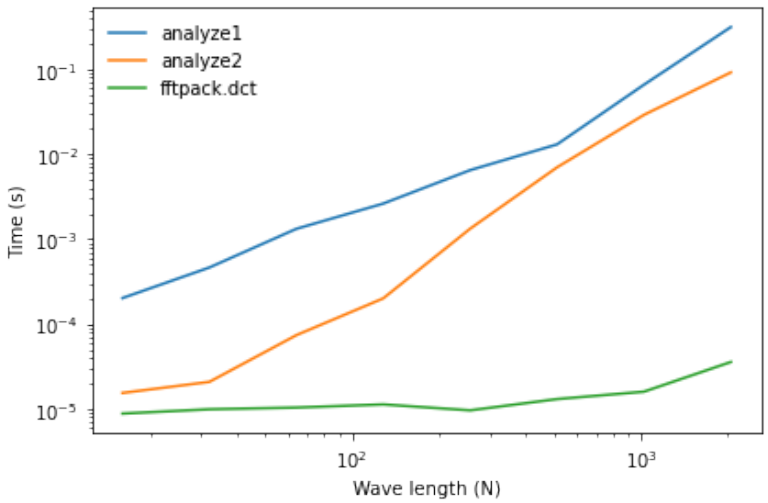


Рис. 1.1: Визуализация данных

У функции fftpack.dct лучшие временные траты.

# Глава 2

# Упражнение 6.2

Берем нашу запис сегмент, а затем ДКП этого сегмента:

```
wave = thinkdsp.read_wave('1008324.wav')
wave.make_audio()
segment = wave.segment(start=2.0, duration=0.5)
segment.normalize()
segment.make_audio()
seg_dct = segment.make_dct()
seg_dct.plot(high=4000)
thinkplot.config(xlabel='Frequency (Hz)', ylabel='DCT')
```

Листинг 2.1: Загрузка звука и сегмент

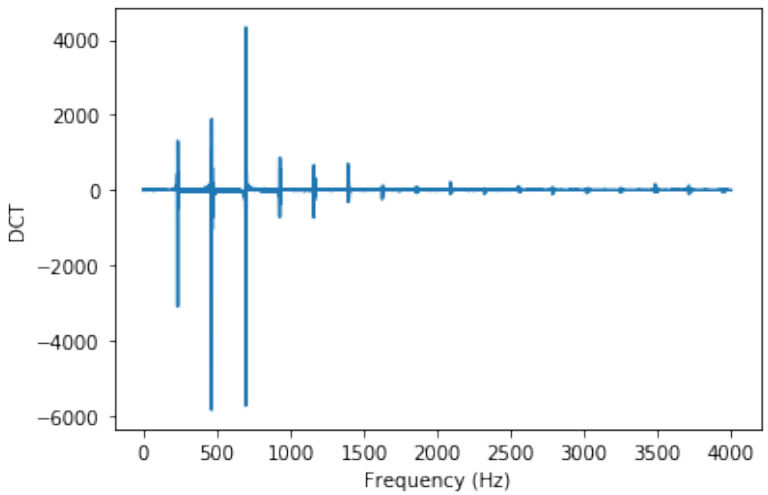


Рис. 2.1: Визуализация сжатого звука

Многие записи близки к нулю. Следующая функция принимает ДКП и устанавливает для элементов ниже порога значение 0.

```
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')
```

Листинг 2.2: Функция `compress`

Если мы применим его к сегменту, мы можем удалить более 90% элементов:

```
seg_dct = segment.make_dct()
compress(seg_dct, thresh=10)
seg_dct.plot(high=4000)
```

Листинг 2.3: Сжатие звука

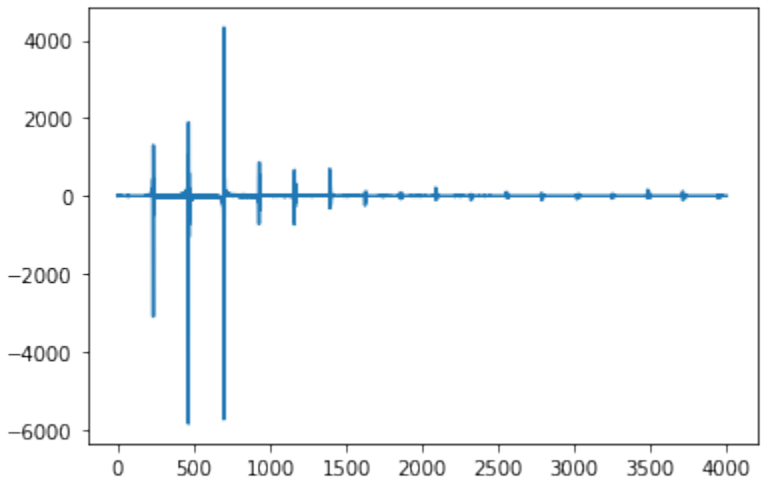


Рис. 2.2: Визуализация сжатого звука

```
seg2 = seg_dct.make_wave()
seg2.make_audio()
```

Листинг 2.4: Воспроизведение сжатого звука

Результат звучит точно так же.

Чтобы сжать более длинный сегмент, мы можем сделать спектрограмму ДКП.

```
def make_dct_spectrogram(wave, seg_length):
    """Computes the DCT spectrogram of the wave.

    seg_length: number of samples in each segment

    returns: Spectrogram
    """
    window = np.hamming(seg_length)
    i, j = 0, seg_length
    step = seg_length // 2

    # map from time to Spectrum
    spec_map = {}

    while j < len(wave.ys):
        segment = wave.slice(i, j)
        segment.window(window)

        # the nominal time for this segment is the midpoint
        t = (segment.start + segment.end) / 2
        spec_map[t] = segment.make_dct()

        i += step
        j += step

    return thinkdsp.Spectrogram(spec_map, seg_length)
```

Листинг 2.5: Функция `make_dct_spectrogram`

Теперь мы можем составить DCT-спектрограмму и применить сжатие к каждому сегменту:

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

Листинг 2.6: Сжатие звука

В большинстве сегментов сжатие составляет 75-80%. Чтобы услышать, как это звучит, мы можем преобразовать спектрограмму обратно в волну и воспроизвести её.

```
wave2 = spectro.make_wave()
wave2.make_audio()
```

Листинг 2.7: Воспроизведение сжатого звука

Так же прослушаем оригинал для сравнения.

```
wave.make_audio()
```

Листинг 2.8: Воспроизведение оригинального звука

При сжатии слышно характерный треск во время воспроизведения аудио, так что можно смело сказать, что нам удалось сжать аудиозапись.

# Глава 3

# Упражнение 6.3

Теперь нам нужно запустить готовый `phase.ipynb` и посмотреть, что там происходит.

Если мы посмотрим на фазовые сдвиги каждой компоненты некоторого звука, то мы будем видеть только много случайных значений, однако если отфильтровать частоты с маленьким амплитудами, то начнет вырисовываться струкутра. Величина фазы от частоты компоненты может зависеть как линейно, так и случайно, однако в подавляющем большинстве случаев ухо не будет способно это воспринять.

# Глава 4

# Выводы

Во время выполнения лабораторной работы получены навыки работы с прямым и обратным дискретным косинусным преобразованием. Также получены навыки их практического применения.