Лабораторная работа 6

Чернышев Ярослав

31 мая 2021 г.

Оглавление

Глава 1

Задание 6.1

В данном задании требуется проверить тезис, что analyze1 отнимает $O(x^3)$ времени, а analyze2 - $O(x^2)$. Сначала напишем функцию, которая будет осуществлять временную проверку:

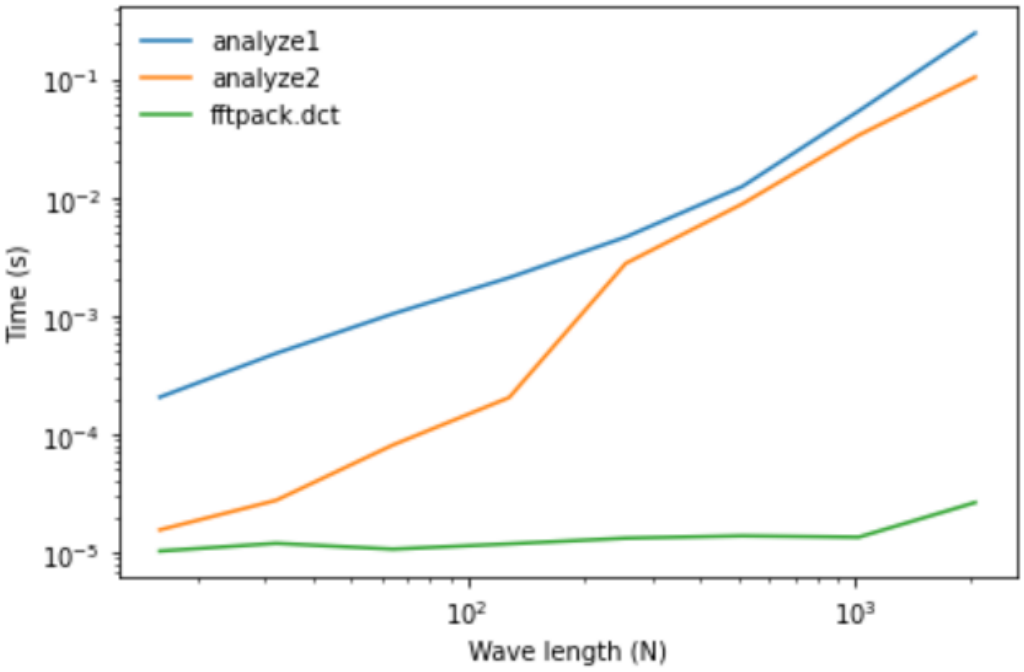
```
def run_speed_test(ns, func):
    results = []
    for N in ns:
    print(N)
    ts = (0.5 + np.arange(N)) / N
    freqs = (0.5 + np.arange(N)) / 2
    ys = noise.ys[:N]
    result = get_ipython().magic('timeit -r1 -o func(ys,
   freqs, ts)')
    results.append(result)
    bests = [result.best for result in results]
    return bests

ns = 2 ** np.arange(6, 13)
bests = run_speed_test(ns, analyze1)
bests2 = run_speed_test(ns, analyze2)
bests3 = run_speed_test(ns, scipy_dct)

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)', **
   loglog)

```

Полученный график:



Можно видеть, что тезис из задания неверен для первой анализирующей функции, но верен для второй. Кроме этого, у функции fftpack.dct лучшие временные затраты.

Глава 2

Задание 6.2

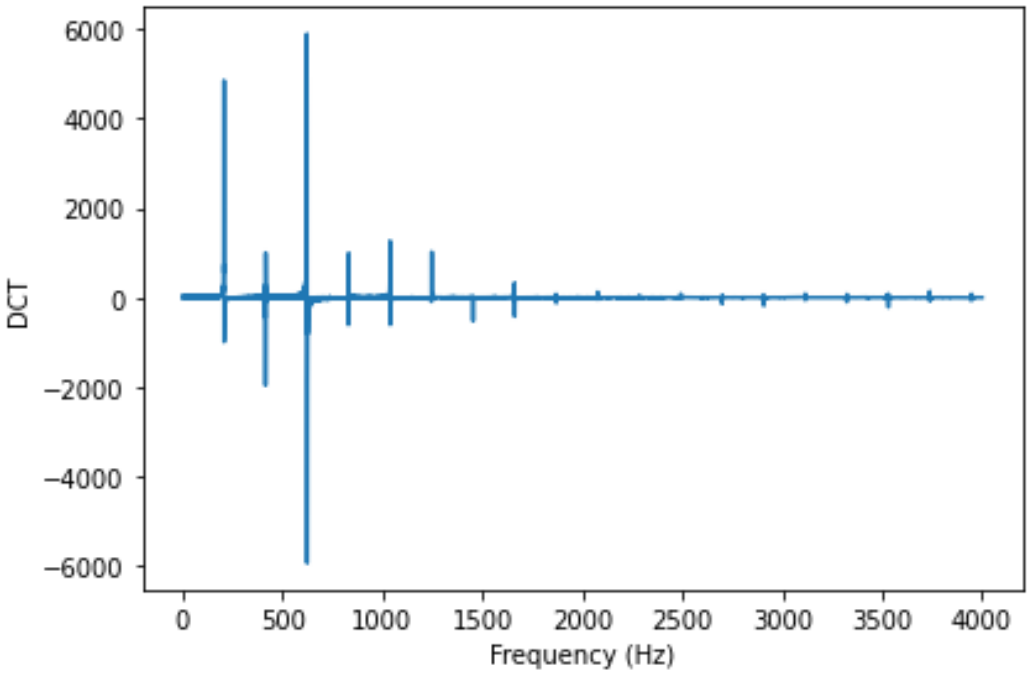
В этом задании требуется поэксперементировать с DCT архивацией.

В качестве "подопытного"взята запись саксофона из предыдущей лабораторной работы. Напишем требуемую подпрограмму и протестируем её:

```
wave = thinkdsp.read_wave('100475F_iluppaiF_saxophone-weep.wav')
segment = wave.segment(start=2.0, duration=0.5)
segment.normalize()
seg_dct = segment.make_dct()
seg_dct.plot(high=4000)
thinkplot.config(xlabel='Frequency (Hz)', ylabel='DCT')

```

Был получен следующий график:



Ниже приведен функция и пример её прменения для обнуления "несущественных"частот:

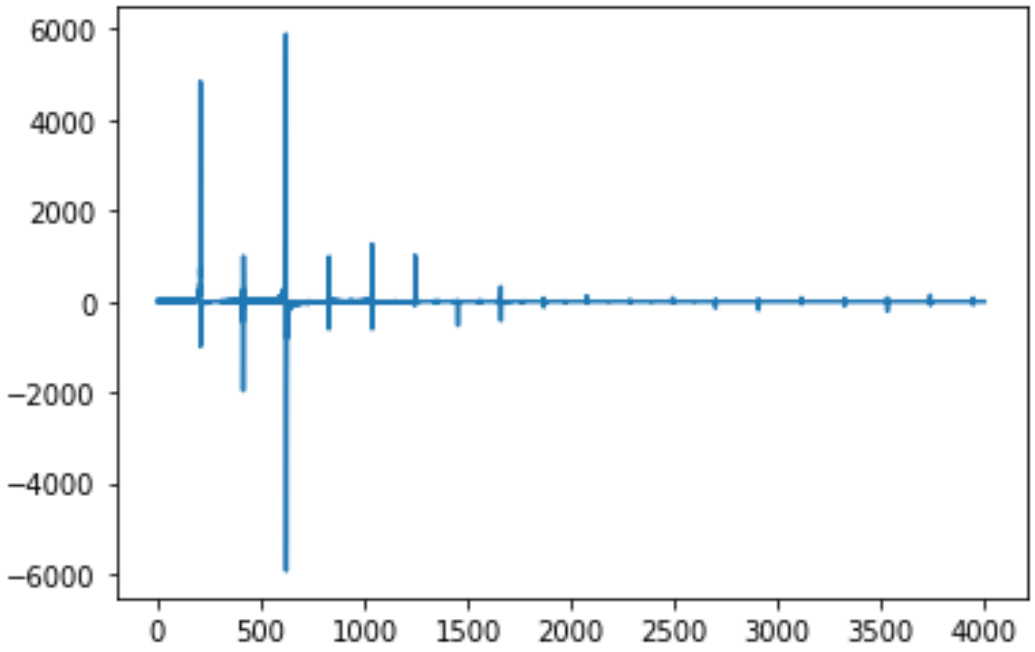
```
def compress(dct, thresh=1):
    count = 0
    for i, amp in enumerate(dct.amps):
        if np.abs(amp) < thresh:
            dct.hs[i] = 0
            count += 1

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

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

```

На слух, "урезанная"версия мало чем отличается от оригинальной. Визуально, тоже:



Для сжатия более длинных фрагментов можно применить функцию, создающую спектрограмму с использованием ДКП:

```
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)

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

```
compress(dct, thresh=0.2)
```

Вывод показал, что удалось избавиться от почти 70-90 % элементов, причем повторное прослушивание показывает, что на звучании это практически никак не сказалось.

Глава 3

Задание 6.3

Согласно заданию в книге мной были проанализированы примеры из файла phase.ipynb для разных сегментов звука. В качестве вывода: удаление компонентов звука, которые не играют большой роли, делает структуру сигнала более читаемой, причем зависимость величины фазы компоненты от её частоты практически не различима на слух, будь она линейна или случайна. В то же время человек способен воспринять разницу между оригиналом и звуком, у которого отсутствуют важные частотные компоненты.