Лабораторная работа 6

Луняк Николай

11 апреля 2021 г.

Оглавление

Список иллюстраций

Листинги

Глава 1

Асимптотика

Нужно оценить асимптотику `analyze1`, `analyze2` и `fftpack.dct`. Чтобы это сделать, подадим на вход `scipy.stats.linregress` логарифмированные значения размера текущей выборки и затраченного времени (потому что, например, $x \to x^k \Rightarrow \log(x) \to k \cdot \log(x)$).

Сначала импортируем накопленные человечеством знания:

```
from thinkdsp import Signal, Sinusoid, SquareSignal,
    TriangleSignal, SawtoothSignal, ParabolicSignal
from thinkdsp import normalize, unbias, PI2, decorate
from thinkdsp import Chirp
from thinkdsp import read_wave
from thinkdsp import Spectrum, Wave,
    UncorrelatedGaussianNoise

import numpy as np
import pandas as pd

from matplotlib import pyplot

import thinkstats2

from scipy.stats import linregress

import scipy
import scipy.fftpack

def analyze1(ys, fs, ts):
    """Analyze a mixture of cosines and return amplitudes.

    Works for the general case where M is not orthogonal.

    ys: wave array
    fs: frequencies in Hz
```

```
    ts: times where the signal was evaluated

    returns: vector of amplitudes
    """
    args = np.outer(ts, fs)
    M = np.cos(PI2 * args)
    amps = np.linalg.solve(M, ys)
    return amps

def analyze2(ys, fs, ts):
    """Analyze a mixture of cosines and return amplitudes.

    Assumes that fs and ts are chosen so that M is orthogonal
    .

    ys: wave array
    fs: frequencies in Hz
    ts: times where the signal was evaluated

    returns: vector of amplitudes
    """
    args = np.outer(ts, fs)
    M = np.cos(PI2 * args)
    amps = np.dot(M, ys) / 2
    return amps

def scipy_dct(ys, freqs, ts):
    return scipy.fftpack.dct(ys, type=3)

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

PI2 = np.pi * 2
```

Листинг 1.1: Импорты

А теперь создадим список проверяемых размеров входных данных, посчитаем время и отобразим все вместе на одном графике.

```
def run_speed_test(counts, code, noise):
    results = []

    for count in counts:
        print(f'For {count} samples:')
        ts = (0.5 + np.arange(count)) / count
        freqs = (0.5 + np.arange(count)) / 2
        ys = noise.ys[:count]
        result = %timeit -r1 -o code(ys, freqs, ts)
        results.append(result)

    return [result.best for result in results]
```

```
13
14 def fit_slope(counts, results):
15     x = np.log(counts)
16     y = np.log(results)
17     return linregress(x, y).slope
18
19 signal = UncorrelatedGaussianNoise()
20 noise = signal.make_wave(duration=1.0, framerate=16384)
21
22 print('Testing analyze1...')
23 counts = 2 ** np.arange(6, 13)
24 results1 = run_speed_test(counts, analyze1, noise)
25 slope1 = fit_slope(counts, results1)
26 print('')
27
28 print('Testing analyze2...')
29 results2 = run_speed_test(counts, analyze2, noise)
30 slope2 = fit_slope(counts, results2)
31 print('')
32
33 print('Testing scipy_dct...')
34 results3 = run_speed_test(counts, scipy_dct, noise)
35 slope3 = fit_slope(counts, results3)
36
37 pyplot.plot(counts, results1, label=f'analyze1 (slope: {
    slope1})')
38 pyplot.plot(counts, results2, label=f'analyze2 (slope: {
    slope2})')
39 pyplot.plot(counts, results3, label=f'fftpack.dct (slope: {
    slope3})')
40 decorate(xlabel='Wave length (N)', ylabel='Time (s)', **
    loglog)
```

Листинг 1.2: Замеры

```
1 Testing analyze1...
2 For 64 samples:
3 2 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops
    each)
4 For 128 samples:
5 4.41 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100
    loops each)
6 For 256 samples:
7 13.7 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100
    loops each)
8 For 512 samples:
9 18.5 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 10
    loops each)
10 For 1024 samples:
```

```
62.2 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 10 loops each)
For 2048 samples:
253 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
For 4096 samples:
1.64 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

Testing analyze2...
For 64 samples:
74.3 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 10000 loops each)
For 128 samples:
479 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 1000 loops each)
For 256 samples:
1.97 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1000 loops each)
For 512 samples:
5.46 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
For 1024 samples:
19.6 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
For 2048 samples:
74.1 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 10 loops each)
For 4096 samples:
287 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)

Testing scipy_dct...
For 64 samples:
9.48 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100000 loops each)
For 128 samples:
9.93 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100000 loops each)
For 256 samples:
10.8 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100000 loops each)
For 512 samples:
12.9 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100000 loops each)
For 1024 samples:
16.5 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 100000 loops each)
For 2048 samples:
```

```
25.8 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 10000
    loops each)
For 4096 samples:
53.7 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 10000
    loops each)
```

Листинг 1.3: Лог

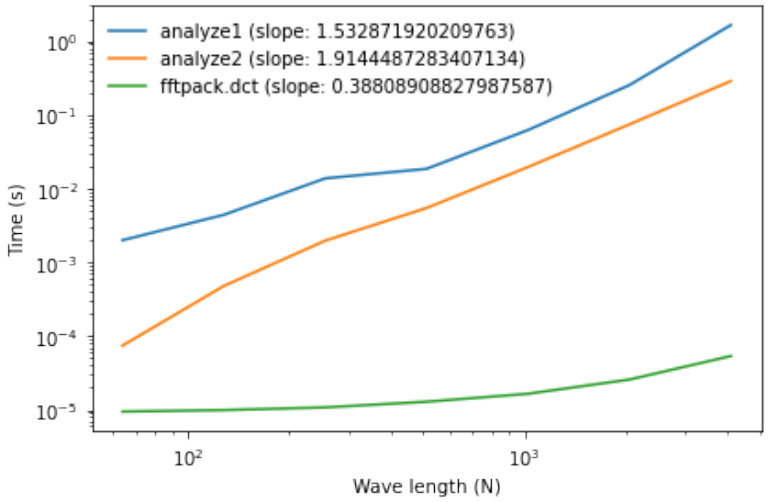


Рис. 1.1: Сравнение

Похоже, что `analyze1` не получилось оценить (возможно, из-за малых размеров выборки), а `analyze2` дает ожидаемое значение наклона.

Последняя функция, `ffpack.dct`, оказывается заметно более быстрой по времени, потому что ее сложность пропорциональна $n \log(n)$.
`analyze1`

Глава 2

Сжатие

Сначаал реализуем простое сжатие для одного небольшого сегмента некоторого звука.

```
def compress(dct, threshold=1, log=False):
    count = 0

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

    total = len(dct.amps)

    if log:
        print(f'Total: {total}, Removed: {count} = {100 *
    count / total:.1f}%', sep='\t')

    return 100 * count / total
```

Листинг 2.1: Сжатие сегмента

Тут просто происходит зануление компонент со «слишком» малыми амплитудами.

Проверим его на некотором звуке.

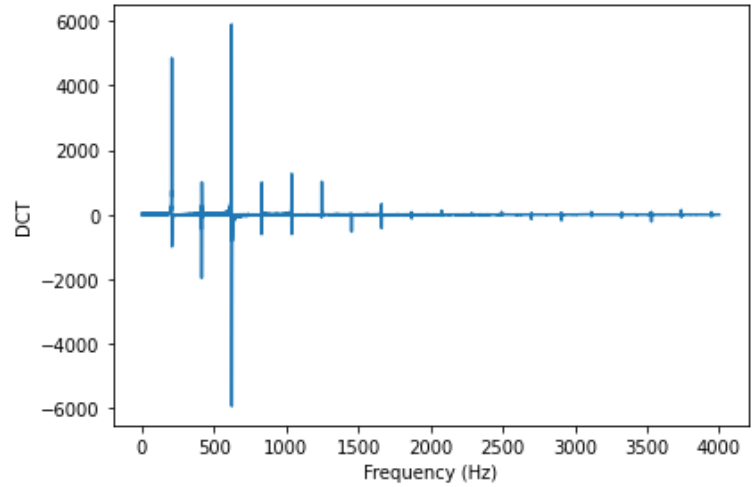
```
wave = read_wave('Sounds/100475__iluppai__saxophone-weep.wav'
    )
segment = wave.segment(start=1.2, duration=0.5)
segment.normalize()

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

pyplot.show()
```

```
seg_dct = segment.make_dct()
compress(seg_dct, threshold=100, log=True)
seg_dct.plot(high=4000)
```

Листинг 2.2: Сжатие сегмента




```
Total: 22050, Removed: 21919 = 99.4%
```

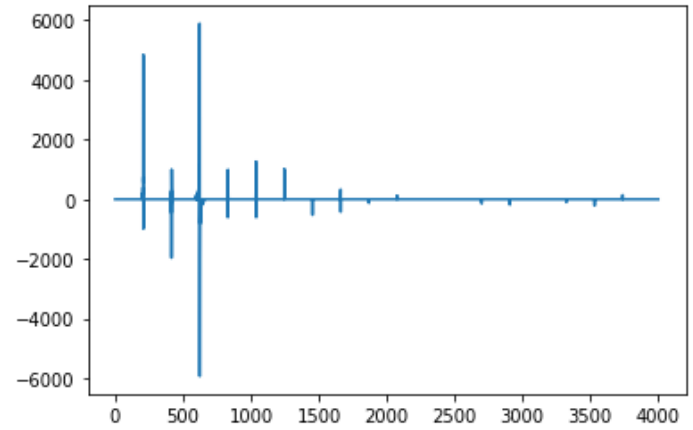


Рис. 2.1: Спектр «до» и «после»

Чтобы сжать длинный изменяющийся во времени звук, нам необходимо брать его спектры в течение некоторых сегментов по аналогии с тем, как это делается для спектрограмм. Отсюда следует, что удобно реализовать себе DCT-спектрограмму.

```
def make_dct_spectrogram(wave, segment_length):
    window = np.hamming(segment_length)
```

```
    i, j = 0, segment_length
    step = segment_length // 2
    spectrums = {}

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

        t = (segment.start + segment.end) / 2
        spectrums[t] = segment.make_dct()

        i += step
        j += step

    return Spectrogram(spectrums, segment_length)

def compress_by_parts(wave, segment_length):
    spectrogram = make_dct_spectrogram(wave, segment_length=
    segment_length)
    average = 0

    for t, dct in sorted(spectrogram.spec_map.items()):
        average += compress(dct, threshold=0.2)

    average /= len(spectrogram.spec_map)

    print(f'Average: {average:.1f}%', sep='\t')

    return spectrogram

wave2 = compress_by_parts(wave, 512).make_wave()
wave2.make_audio()
```

Листинг 2.3: Сжатие длинного звука

После запуска лог дает `Average: 80.9%`. На слух звучит примерно так же.

Глава 3

Влияние фазы

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

Так как вставлять подробные картинки из notebook'а сюда долго, я лишь опишу процесс словами.

Если мы посмотрим на фазовые сдвиги каждой компоненты некоторого звука, то мы будем видеть только «нагромождение» случайных значений, однако если отфильтровать частоты с «незначительными» 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. Отсутствие зависимости и «рандомизацию» уловить еще можно кое-как, но не сдвиг всех компонент по фазе (что в принципе логично, нам не важно, «когда» мы начали слушать звук).

Для звуков с «пропавшей» частотой наблюдается особенность: фазовую структуру таких звуков ухо может воспринимать, однако автор предположил, что это связано с тем, что мозг «пытается» учесть автокорреляцию.