

Лабораторная работа №6
Дискретное косинусное преобразование

Кобыжев Александр

10 апреля 2021 г.

Оглавление

1	Упражнение 6.1	5
2	Упражнение 6.2	11
3	Упражнение 6.3	15
3.1	Пилообразная волна	15
3.2	Запись габоя	21
3.3	Запись саксофона	23
3.4	Урезание фундаментальных гармоник	26
3.5	Построение АКФ	28
4	Выводы	31

Список иллюстраций

1.1	Визуализация данных с <code>analyze1</code>	7
1.2	Визуализация данных с <code>analyze2</code>	8
1.3	Визуализация данных с <code>scipy.fftpack.dct</code>	9
1.4	Визуализация всех кривых	10
2.1	Визуализация сжатого звука	12
2.2	Визуализация сжатого звука	13
3.1	Визуализация сигнала	16
3.2	Спектр сигнала	16
3.3	Визуализация угловой части спектра с <code>tresh = 0</code>	17
3.4	Визуализация угловой части спектра с <code>tresh = 1</code>	18
3.5	Визуализация неизменённого спектра	19
3.6	Визуализация при нулевых углах	19
3.7	Визуализация при повороте углов	20
3.8	Визуализация при случайных углах	21
3.9	Визуализация неизменённого звука	22
3.10	Визуализация при нулевых углах	22
3.11	Визуализация при повороте углов	23
3.12	Визуализация при случайных углах	23
3.13	Визуализация неизменённого звука	24
3.14	Визуализация при нулевых углах	24
3.15	Визуализация при повороте углов	25
3.16	Визуализация при случайных углах	25
3.17	Урезание частот звука	26
3.18	Визуализация неизменённого звука	27
3.19	Визуализация при нулевых углах	27
3.20	Визуализация при повороте углов	28
3.21	Визуализация при случайных углах	28
3.22	Построение АКФ	29

Листинги

1.1	Создание сигнала	5
1.2	Массив степеней двойки	5
1.3	Функция <code>plot_bests</code>	5
1.4	Работа с <code>analyze1</code>	5
1.5	Результат работы	6
1.6	Работа с <code>analyze2</code>	7
1.7	Результат работы	7
1.8	Работа с <code>scipy.fftpack.dct</code>	8
1.9	Результат работы	8
1.10	Визуализация всех кривых	10
2.1	Загрузка звука	11
2.2	Сегмент звука	11
2.3	Сжатие звука	11
2.4	Функция <code>compress</code>	12
2.5	Сжатие звука	12
2.6	Воспроизведение сжатого звука	13
2.7	Функция <code>make_dct_spectrogram</code>	13
2.8	Сжатие звука	14
2.9	Воспроизведение сжатого звука	14
2.10	Воспроизведение оригинального звука	14
3.1	Создание пилообразного сигнала	15
3.2	Визуализация сигнала	15
3.3	Спектр сигнала	16
3.4	Функция <code>plot_angle</code>	16
3.5	Визуализация угловой части спектра с <code>tresh</code>	17
3.6	Визуализация угловой части спектра с <code>tresh</code>	17
3.7	Функция <code>plot_three</code>	18
3.8	Визуализация неизменённого спектра	18
3.9	Функция <code>zero_angle</code>	19
3.10	Визуализация при нулевых углах	19
3.11	Функция <code>rotate_angle</code>	20

3.12	Визуализация при повороте углов	20
3.13	Функция <code>random_angle</code>	20
3.14	Визуализация при случайных углах	21
3.15	Загрузка звука	21
3.16	Визуализация неизменённого звука	21
3.17	Визуализация при нулевых углах	22
3.18	Визуализация при повороте углов	22
3.19	Визуализация при случайных углах	23
3.20	Загрузка звука	24
3.21	Визуализация неизменённого звука	24
3.22	Визуализация при нулевых углах	24
3.23	Визуализация при повороте углов	25
3.24	Визуализация при случайных углах	25
3.25	Урезание частот звука	26
3.26	Визуализация неизменённого звука	26
3.27	Визуализация при нулевых углах	27
3.28	Визуализация при повороте углов	27
3.29	Визуализация при случайных углах	28
3.30	Функция <code>autocorr</code>	29
3.31	Функция <code>plot_acf</code>	29
3.32	Построение АКФ	29

Глава 1

Упражнение 6.1

Начнём с шумового сигнала и массива величин степени двойки:

```
1 signal = thinkdsp.UncorrelatedGaussianNoise()
2 noise = signal.make_wave(duration=1.0, framerate=16384)
3 noise.ys.shape
```

Листинг 1.1: Создание сигнала

Shape сигнала составляет 16384.

```
1 ns = 2 ** np.arange(6, 15)
2 ns
```

Листинг 1.2: Массив степеней двойки

Следующая функция берёт массив результатов временного эксперимента, отображает результаты и выстраивает прямую линию.

```
1 def plot_bests(bests):
2     thinkplot.plot(ns, bests)
3     thinkplot.config(xscale='log', yscale='log', legend=False)
4
5     x = np.log(ns)
6     y = np.log(bests)
7     t = scipy.stats.linregress(x,y)
8     slope = t[0]
9
10    return slope
```

Листинг 1.3: Функция plot_bests

Рассмотрим результаты для `analyze1`.

```
1 results = []
```

```

2 for N in ns:
3     print(N)
4     ts = (0.5 + np.arange(N)) / N
5     freqs = (0.5 + np.arange(N)) / 2
6     ys = noise.ys[:N]
7     result = %timeit -r1 -o dct.analyze1(ys, freqs, ts)
8     results.append(result)
9
10 bests = [result.best for result in results]
11 plot_bests(bests)

```

Листинг 1.4: Работа с analyze1

```

1 64
2 657 µs ± 0 ns per loop (mean ± std. dev. of 1 run, 1000 loops each)
3 128
4 1.42 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1000 loops each)
5 256
6 3.64 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
7 512
8 9.54 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
9 1024
10 33.3 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 10 loops each)
11 2048
12 152 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 10 loops each)
13 4096
14 676 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
15 8192
16 4.19 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
17 16384
18 25.7 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
19
20 1.9111873370989283

```

Листинг 1.5: Результат работы

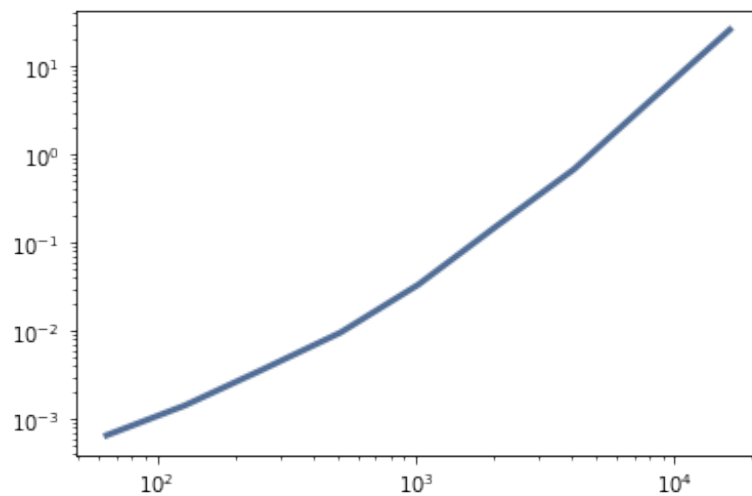


Рис. 1.1: Визуализация данных с `analyze1`

Наклон близок к 2, а не к 3. Одна из причин состоит в том, что производительность `np.linalg.solve` почти квадратична в этом диапазоне размеров массива.

Линия изогнута, что говорит нам о том, что мы не достигли размера массива, при котором среда выполнения показывает кубический рост. Я думаю, что при больших размерах массива наклон в конечном итоге сходится к 3.

Теперь рассмотрим результат для `analyze2`:

```

1 results = []
2 for N in ns:
3     ts = (0.5 + np.arange(N)) / N
4     freqs = (0.5 + np.arange(N)) / 2
5     ys = noise.ys[:N]
6     result = %timeit -r1 -o dct.analyze2(ys, freqs, ts)
7     results.append(result)
8
9 bests2 = [result.best for result in results]
10 plot_bests(bests2)
```

Листинг 1.6: Работа с `analyze2`

```

1 42.5 μs ± 0 ns per loop (mean ± std. dev. of 1 run, 10000 loops each)
2 212 μs ± 0 ns per loop (mean ± std. dev. of 1 run, 1000 loops each)
3 1.04 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1000 loops each)
4 3.22 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
```



```

5 12.2 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 100 loops each)
6 47.9 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 10 loops each)
7 189 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
8 722 ms ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
9 3.05 s ± 0 ns per loop (mean ± std. dev. of 1 run, 1 loop each)
10
11 1.9772047966153776

```

Листинг 1.7: Результат работы

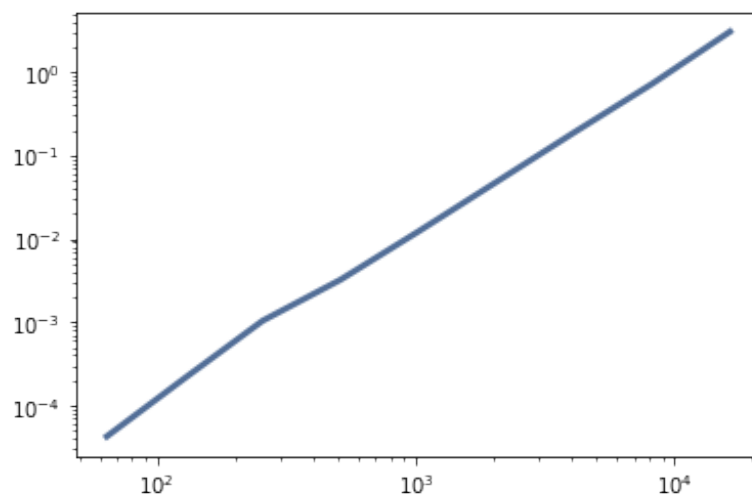


Рис. 1.2: Визуализация данных с `analyze2`

Результаты для `analyze2`, как и ожидалось, складываются в прямую линию с предполагаемым наклоном, близким к 2.

Рассмотрим результаты для `scipy.fftpack.dct`:

```

1 results = []
2 for N in ns:
3     ys = noise.ys[:N]
4     result = %timeit -o scipy.fftpack.dct(ys, type=3)
5     results.append(result)
6
7 bests3 = [result.best for result in results]
8 plot_bests(bests3)

```

Листинг 1.8: Работа с `scipy.fftpack.dct`

```

1 5.33 µs ± 538 ns per loop (mean ± std. dev. of 7 runs, 100000 loops
  each)

```

```

2 5.25  $\mu$ s  $\pm$ 171 ns per loop (mean  $\pm$ std. dev. of 7 runs, 100000 loops
   each)
3 6.55  $\mu$ s  $\pm$ 408 ns per loop (mean  $\pm$ std. dev. of 7 runs, 100000 loops
   each)
4 7.7  $\mu$ s  $\pm$ 128 ns per loop (mean  $\pm$ std. dev. of 7 runs, 100000 loops
   each)
5 9.5  $\mu$ s  $\pm$ 339 ns per loop (mean  $\pm$ std. dev. of 7 runs, 100000 loops
   each)
6 14.9  $\mu$ s  $\pm$ 407 ns per loop (mean  $\pm$ std. dev. of 7 runs, 100000 loops
   each)
7 27.3  $\mu$ s  $\pm$ 141 ns per loop (mean  $\pm$ std. dev. of 7 runs, 10000 loops
   each)
8 54.5  $\mu$ s  $\pm$ 470 ns per loop (mean  $\pm$ std. dev. of 7 runs, 10000 loops
   each)
9 107  $\mu$ s  $\pm$ 797 ns per loop (mean  $\pm$ std. dev. of 7 runs, 10000 loops
   each)
10
11 0.5564395911984347

```

Листинг 1.9: Результат работы

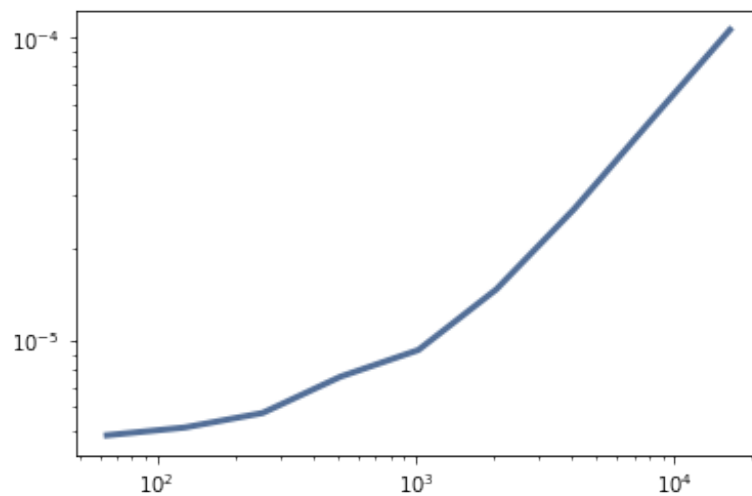


Рис. 1.3: Визуализация данных с `scipy.fftpack.dct`

Эта реализация `dct` ещё быстрее. Линия изогнута, что означает, что либо мы ещё не видели асимптотическое поведение, либо асимптотическое поведение не является простым показателем n . Фактически, как мы скоро увидим, время выполнения пропорционально $n \log n$.

Теперь рассмотрим все кривые на одних осях:

```

1 thinkplot.preplot(3)
2 thinkplot.plot(ns, bests, label='analyze1')
3 thinkplot.plot(ns, bests2, label='analyze2')
4 thinkplot.plot(ns, bests3, label='fftpack.dct')
5 thinkplot.config(xscale='log', yscale='log', legend=True,
    loc='upper left')

```

Листинг 1.10: Визуализация всех кривых

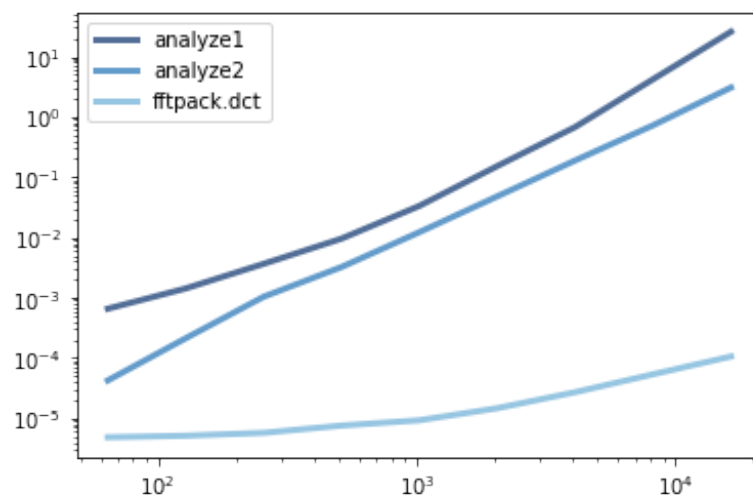


Рис. 1.4: Визуализация всех кривых

Глава 2

Упражнение 6.2

`thinkdsp` предоставляет класс `Dct`, похожий на `Spectrum`, но использующий DCT вместо FFT.

Воспользуемся записью саксофона из предыдущей лабораторной работы:

```
1 wave = thinkdsp.read_wave('100475__iluppai__saxophone-weep.wav')
2 wave.make_audio()
```

Листинг 2.1: Загрузка звука

Теперь сделаем короткий сегмент, а затем ДКП этого сегмента:

```
1 segment = wave.segment(start=2.0, duration=0.5)
2 segment.normalize()
3 segment.make_audio()
```

Листинг 2.2: Сегмент звука

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

Листинг 2.3: Сжатие звука

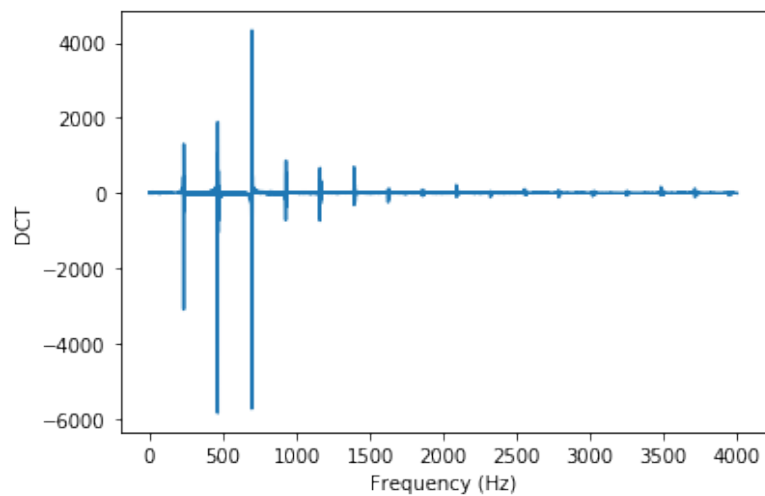


Рис. 2.1: Визуализация сжатого звука

Есть только несколько гармоник со значительной амплитудой, и многие записи близки к нулю. Следующая функция принимает ДКП и устанавливает для элементов ниже порога значение 0.

```

1 def compress(dct, thresh=1):
2     count = 0
3     for i, amp in enumerate(dct.amps):
4         if abs(amp) < thresh:
5             dct.hs[i] = 0
6             count += 1
7
8     n = len(dct.amps)
9     print(count, n, 100 * count / n, sep='\t')
```

Листинг 2.4: Функция `compress`

Если мы применим его к сегменту, мы можем удалить более 90

```

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

Листинг 2.5: Сжатие звука

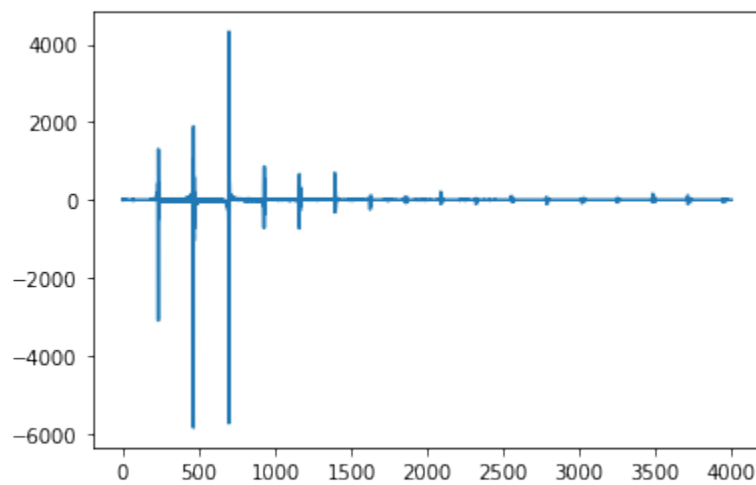


Рис. 2.2: Визуализация сжатого звука

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

Листинг 2.6: Воспроизведение сжатого звука

Результат звучит точно так же, по ощущениям ничего не изменилось.

Чтобы сжать более длинный сегмент, мы можем сделать спектрограмму ДКП. Следующая функция похожа на `wave.make_spectrogram` за исключением того, что использует ДКП.

```
1 def make_dct_spectrogram(wave, seg_length):
2     """Computes the DCT spectrogram of the wave.
3
4     seg_length: number of samples in each segment
5
6     returns: Spectrogram
7     """
8     window = np.hamming(seg_length)
9     i, j = 0, seg_length
10    step = seg_length // 2
11
12    # map from time to Spectrum
13    spec_map = {}
14
15    while j < len(wave.ys):
16        segment = wave.slice(i, j)
17        segment.window(window)
```

```

18
19     # the nominal time for this segment is the midpoint
20     t = (segment.start + segment.end) / 2
21     spec_map[t] = segment.make_dct()
22
23     i += step
24     j += step
25
26     return thinkdsp.Spectrogram(spec_map, seg_length)

```

Листинг 2.7: Функция `make_dct_spectrogram`

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

```

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

```

Листинг 2.8: Сжатие звука

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

```

1 wave2 = spectro.make_wave()
2 wave2.make_audio()

```

Листинг 2.9: Воспроизведение сжатого звука

Так же прослушаем оригинал для сравнения.

```

1 wave.make_audio()

```

Листинг 2.10: 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

3.1 Пилообразная волна

Для начала воспользуемся пилообразной формой волны, а затем перейдём к более естественным звукам.

```
1 signal = thinkdsp.SawtoothSignal(freq=300, offset=0)
2 wave = signal.make_wave(duration=0.5, framerate=40000)
3 wave.make_audio()
```

Листинг 3.1: Создание пилообразного сигнала

```
1 wave.segment(duration=0.01).plot()
```

Листинг 3.2: Визуализация сигнала

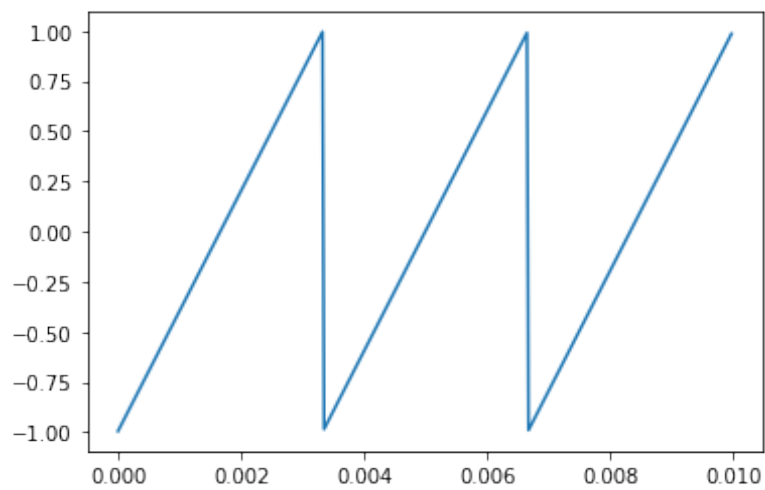


Рис. 3.1: Визуализация сигнала

```

1 spectrum = wave.make_spectrum()
2 spectrum.plot()

```

Листинг 3.3: Спектр сигнала

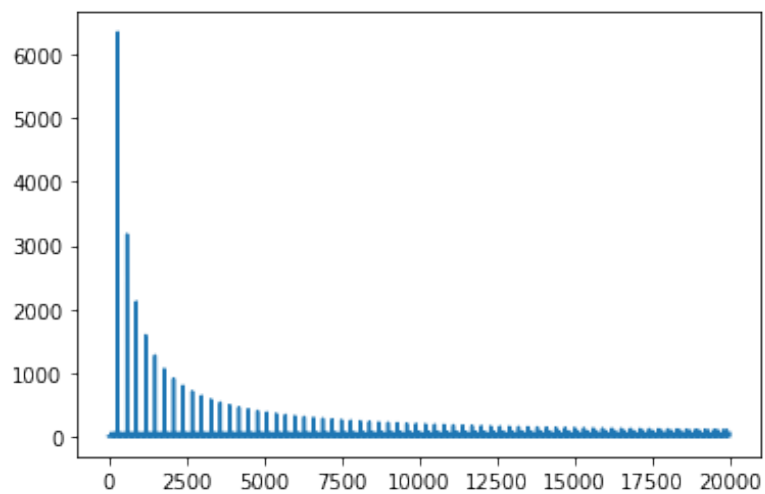


Рис. 3.2: Спектр сигнала

Рассмотрим угловую часть спектра при помощи функции `plot_angle`.

```

1 def plot_angle(spectrum, thresh=1):
2     angles = spectrum.angles

```

```

3 angles[spectrum.amps < thresh] = np.nan
4 thinkplot.plot(spectrum.fs, angles, style='x')
5 thinkplot.config(xlim=[0, spectrum.max_freq],
6                  ylim=[-np.pi, np.pi])

```

Листинг 3.4: Функция `plot_angle`

На большинстве частот амплитуда мала, а угол - в значительной степени случайное число. Так что если мы построим все углы, получится небольшой беспорядок.

```

1 plot_angle(spectrum, thresh=0)
2 thinkplot.config(xlim=[0, spectrum.max_freq], ylim = [-np.pi,
  np.pi])

```

Листинг 3.5: Визуализация угловой части спектра с `thresh`

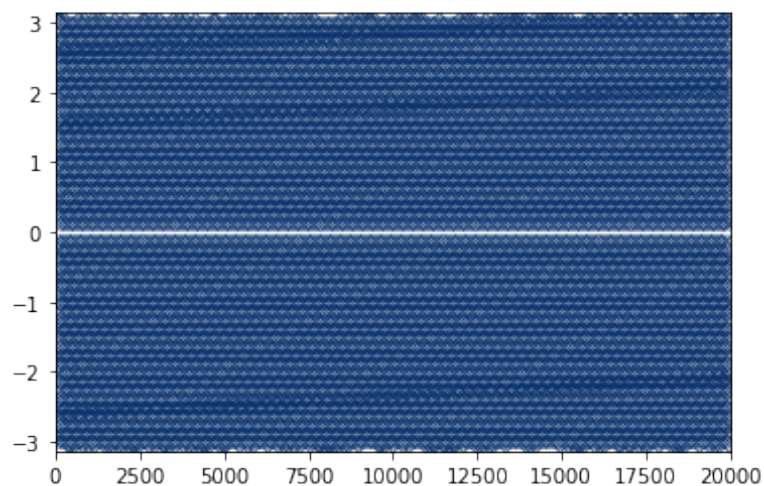


Рис. 3.3: Визуализация угловой части спектра с `thresh = 0`

```

1 plot_angle(spectrum, thresh=1)
2 thinkplot.config(xlim=[0, spectrum.max_freq], ylim = [-np.pi,
  np.pi])

```

Листинг 3.6: Визуализация угловой части спектра с `thresh`

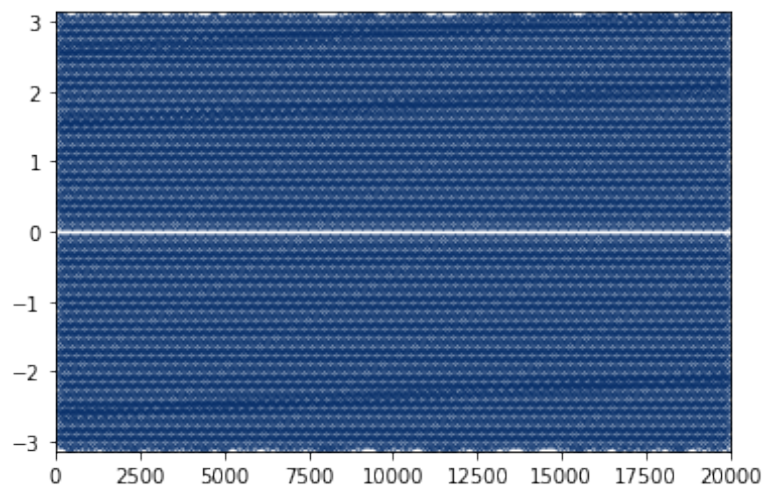


Рис. 3.4: Визуализация угловой части спектра с $\text{thresh} = 1$

Когда мы выбираем только те частоты, где величина превышает пороговое значение, мы видим, что в углах есть структура. Каждая гармоника смещена от предыдущей на доли радиана.

Следующая функция отображает амплитуды, углы и форму волны для заданного спектра.

```

1 def plot_three(spectrum, thresh=1):
2     thinkplot.preplot(cols=3)
3     spectrum.plot()
4     thinkplot.subplot(2)
5     plot_angle(spectrum, thresh=thresh)
6     thinkplot.subplot(3)
7     wave = spectrum.make_wave()
8     wave.segment(duration=0.01).plot()
9     wave.apodize()
10    display(wave.make_audio())

```

Листинг 3.7: Функция `plot_three`

Теперь мы можем визуализировать неизменённый спектр:

```

1 plot_three(spectrum)

```

Листинг 3.8: Визуализация неизменённого спектра

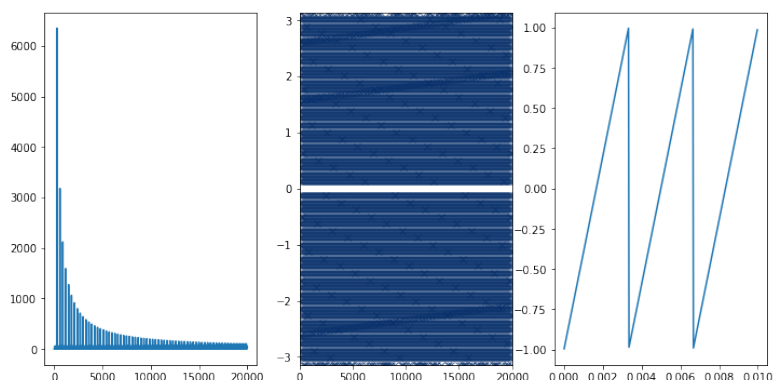


Рис. 3.5: Визуализация неизменённого спектра

Теперь рассмотрим, что произойдёт, если мы установим все углы в ноль.

```

1 def zero_angle(spectrum):
2     res = spectrum.copy()
3     res.hs = res.amps
4     return res

```

Листинг 3.9: Функция `zero_angle`

Амплитуды не изменились, все углы равны нулю, и форма волны выглядит совсем иначе. Но волна звучит примерно так же, только появился какой-то шум, да и громкость стала немного ниже.

```

1 spectrum2 = zero_angle(spectrum)
2 plot_three(spectrum2)

```

Листинг 3.10: Визуализация при нулевых углах

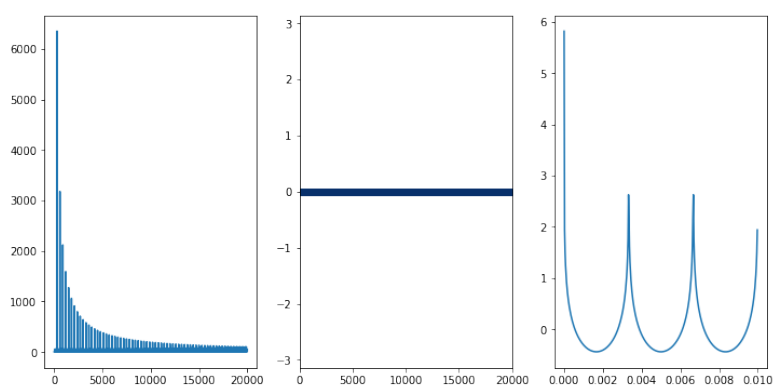


Рис. 3.6: Визуализация при нулевых углах

Если мы умножим комплексные компоненты на $\exp(i\phi)$ это приведет к добавлению ϕ к углам:

```
1 def rotate_angle(spectrum, offset):
2     res = spectrum.copy()
3     res.hs *= np.exp(1j * offset)
4     return res
```

Листинг 3.11: Функция `rotate_angle`

Эффект можно увидеть на рисунке ниже. Опять же, форма волны другая, но звучит примерно так же.

```
1 spectrum3 = rotate_angle(spectrum, 1)
2 plot_three(spectrum3)
```

Листинг 3.12: Визуализация при повороте углов

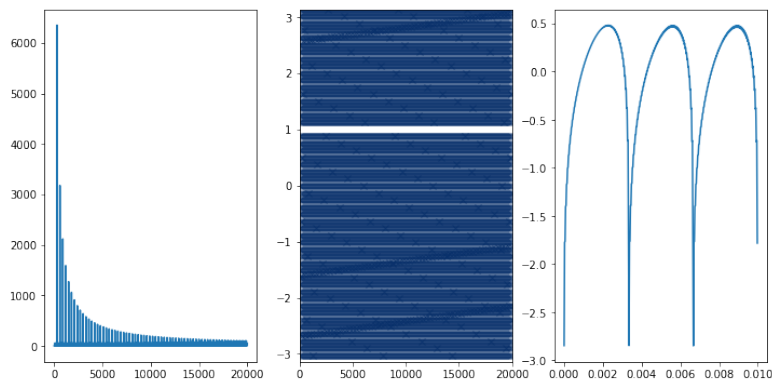


Рис. 3.7: Визуализация при повороте углов

Теперь рассмотрим случай, если мы установим углы на случайные значения.

```
1 PI2 = np.pi * 2
2
3 def random_angle(spectrum):
4     res = spectrum.copy()
5     angles = np.random.uniform(0, PI2, len(spectrum))
6     res.hs *= np.exp(1j * angles)
7     return res
```

Листинг 3.13: Функция `random_angle`

Влияние на форму волны сильное, но воспринимаемый звук остаётся плюс минус прежним, правда он приобрёл "металлический" эффект.

```
1 spectrum4 = random_angle(spectrum)
2 plot_three(spectrum4)
```

Листинг 3.14: Визуализация при случайных углах

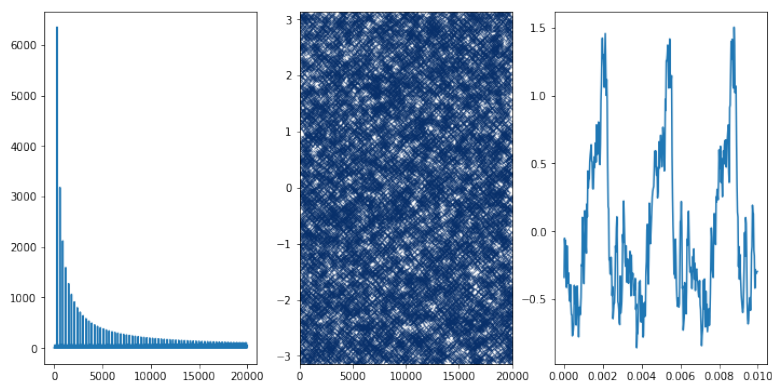


Рис. 3.8: Визуализация при случайных углах

3.2 Запись гобоя

С более естественными звуками результаты несколько отличаются. Рассмотрим запись гобоя.

```
1 wave = thinkdsp.read_wave('120994__thirsk__120-oboe.wav')
2 wave.make_audio()
```

Листинг 3.15: Загрузка звука

```
1 segment = wave.segment(start=0.2, duration=0.6)
2 spectrum = segment.make_spectrum()
3 plot_three(spectrum, thresh=50)
```

Листинг 3.16: Визуализация неизменённого звука

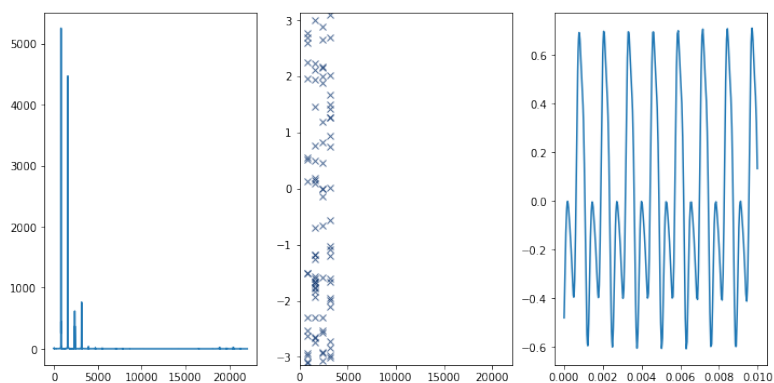


Рис. 3.9: Визуализация неизменённого звука

Здесь все углы установлены в ноль.

```
1 spectrum2 = zero_angle(spectrum)
2 plot_three(spectrum2, thresh=50)
```

Листинг 3.17: Визуализация при нулевых углах

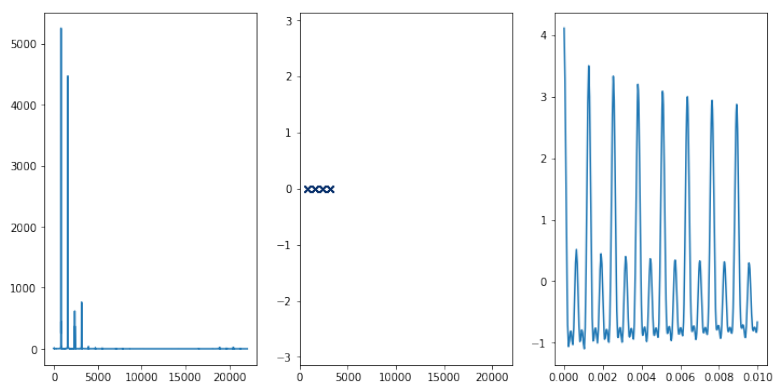


Рис. 3.10: Визуализация при нулевых углах

Теперь рассмотрим случай, когда углы повёрнуты на 1 радиан.

```
1 spectrum3 = rotate_angle(spectrum, 1)
2 plot_three(spectrum3, thresh=50)
```

Листинг 3.18: Визуализация при повороте углов

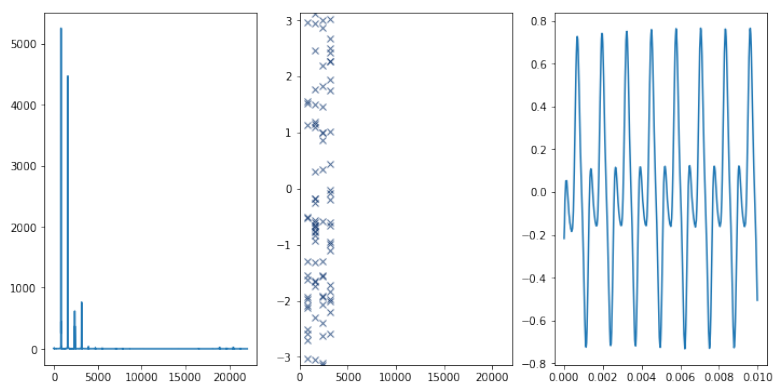


Рис. 3.11: Визуализация при повороте углов

Теперь рассмотрим случайные углы.

```
1 spectrum4 = random_angle(spectrum)
2 plot_three(spectrum4, thresh=50)
```

Листинг 3.19: Визуализация при случайных углах

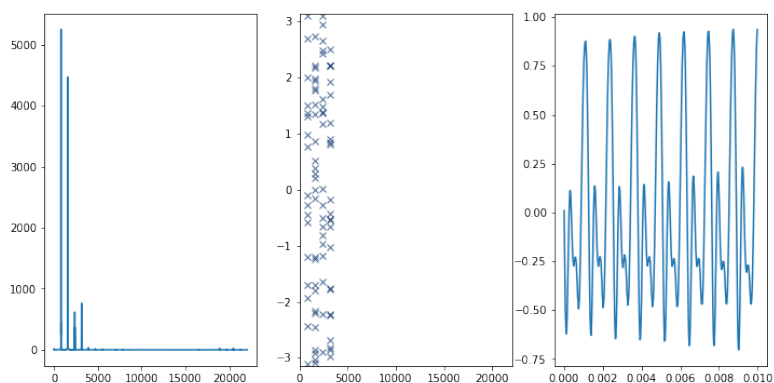


Рис. 3.12: Визуализация при случайных углах

Как мне кажется, установка углов в ноль снижает громкость, поворот углов не имеет вообще никакого эффекта, а случайные углы придают звуку какой-то эффект реверберации.

Попробуем то же самое с отрывком из записи саксофона.

3.3 Запись саксофона


```

1 wave = thinkdsp.read_wave('100475__iluppai__saxophone-weep.wav')
2 wave.make_audio()

```

Листинг 3.20: Загрузка звука

```

1 segment = wave.segment(start=2.5, duration=0.6)
2 spectrum = segment.make_spectrum()
3 plot_three(spectrum, thresh=50)

```

Листинг 3.21: Визуализация неизменённого звука

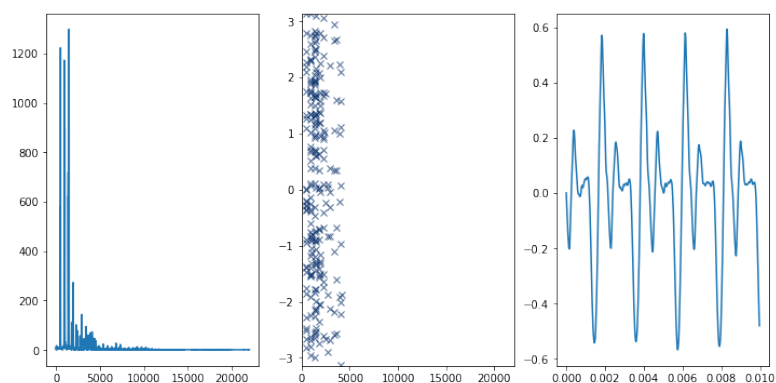


Рис. 3.13: Визуализация неизменённого звука

Здесь все углы установлены в ноль.

```

1 spectrum2 = zero_angle(spectrum)
2 plot_three(spectrum2, thresh=50)

```

Листинг 3.22: Визуализация при нулевых углах

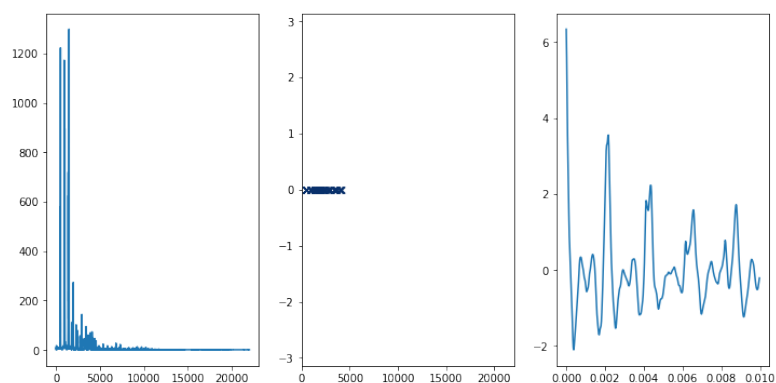


Рис. 3.14: Визуализация при нулевых углах

Теперь рассмотрим случай, когда углы повёрнуты на 1 радиан.

```
1 spectrum3 = rotate_angle(spectrum, 1)
2 plot_three(spectrum3, thresh=50)
```

Листинг 3.23: Визуализация при повороте углов

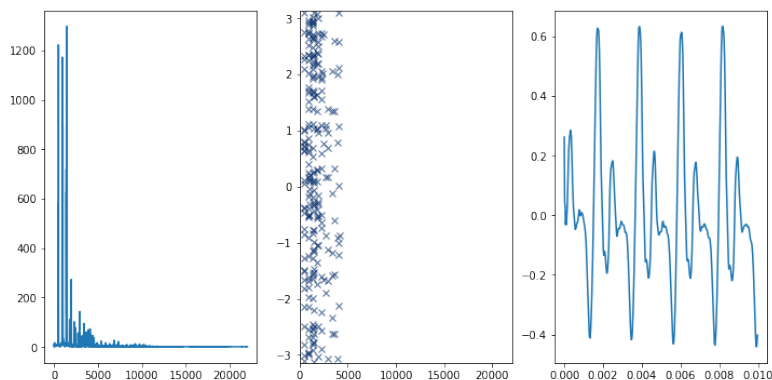


Рис. 3.15: Визуализация при повороте углов

Теперь рассмотрим случайные углы.

```
1 spectrum4 = random_angle(spectrum)
2 plot_three(spectrum4, thresh=50)
```

Листинг 3.24: Визуализация при случайных углах

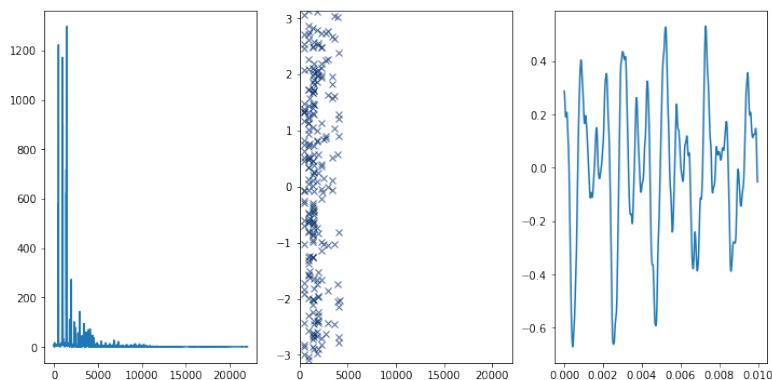


Рис. 3.16: Визуализация при случайных углах

Прослушав все звуки можно сделать вывод, что обнуление делает со звуком какие-то странные вещи, будто бы он стал каким-то "пьяным вра-

щение вновь не имеет никакого эффекта, а случайные углы добавляют вновь эффект реверберации.

Также я знаю, что саксофон отличается от других звуков тем, что основной компонент не является доминирующим. Я предполагаю, что для подобных звуков ухо использует что-то вроде автокорреляции в дополнение к спектральному анализу, и возможно, что этот вторичный режим анализа более чувствителен к фазовой структуре.

Если это так, эффект должен быть более сильным, если фундаментальный вообще отсутствует. Давайте в этом убедимся.

3.4 Урезание фундаментальных гармоник

```
1 spectrum.high_pass(600)
2 spectrum.plot(high=4000)
```

Листинг 3.25: Урезание частот звука

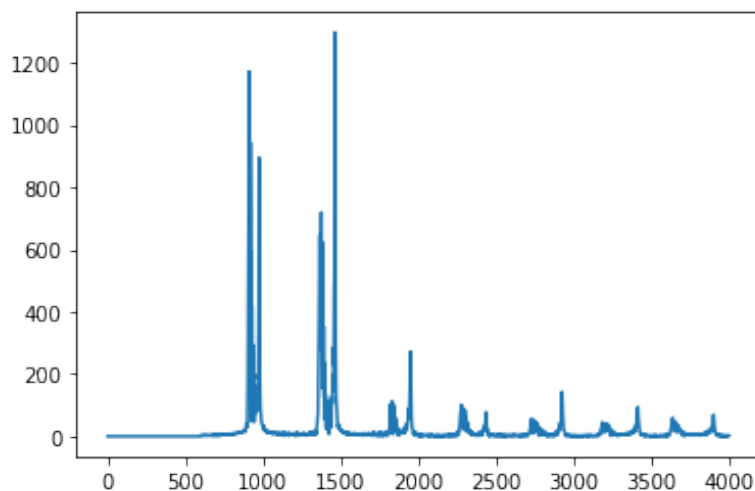


Рис. 3.17: Урезание частот звука

```
1 segment = wave.segment(start=2.5, duration=0.6)
2 spectrum = segment.make_spectrum()
3 plot_three(spectrum, thresh=50)
```

Листинг 3.26: Визуализация неизменённого звука

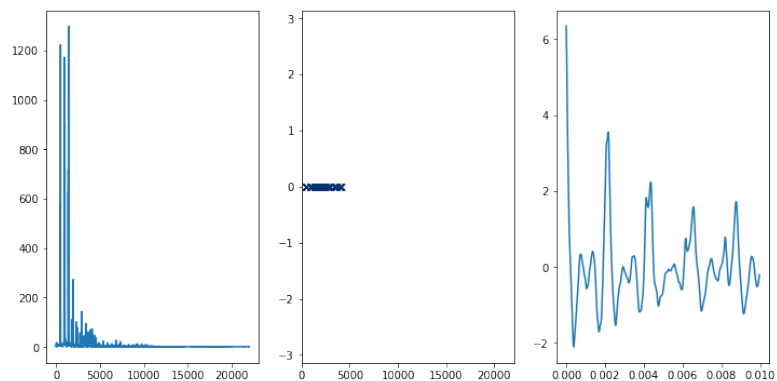


Рис. 3.18: Визуализация неизменённого звука

Здесь все углы установлены в ноль.

```
1 spectrum2 = zero_angle(spectrum)
2 plot_three(spectrum2, thresh=50)
```

Листинг 3.27: Визуализация при нулевых углах

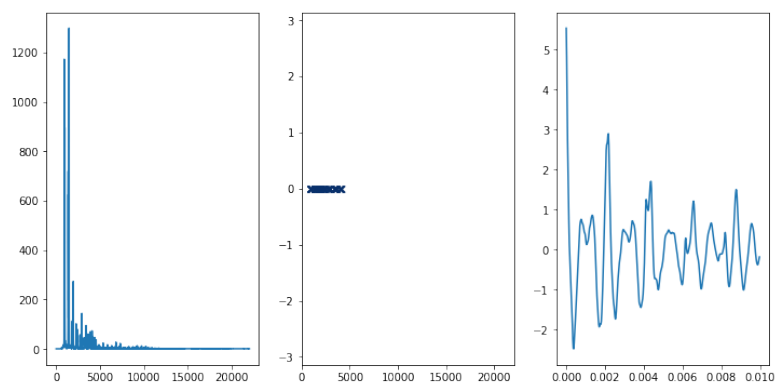


Рис. 3.19: Визуализация при нулевых углах

Теперь рассмотрим случай, когда углы повёрнуты на 1 радиан.

```
1 spectrum3 = rotate_angle(spectrum, 1)
2 plot_three(spectrum3, thresh=50)
```

Листинг 3.28: Визуализация при повороте углов

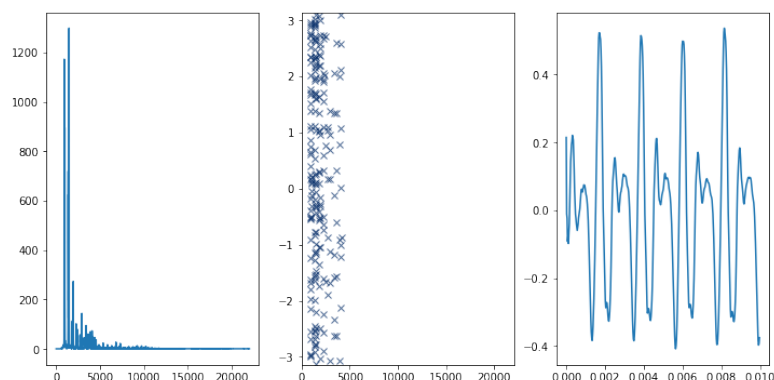


Рис. 3.20: Визуализация при повороте углов

Теперь рассмотрим случайные углы.

```
1 spectrum4 = random_angle(spectrum)
2 plot_three(spectrum4, thresh=50)
```

Листинг 3.29: Визуализация при случайных углах

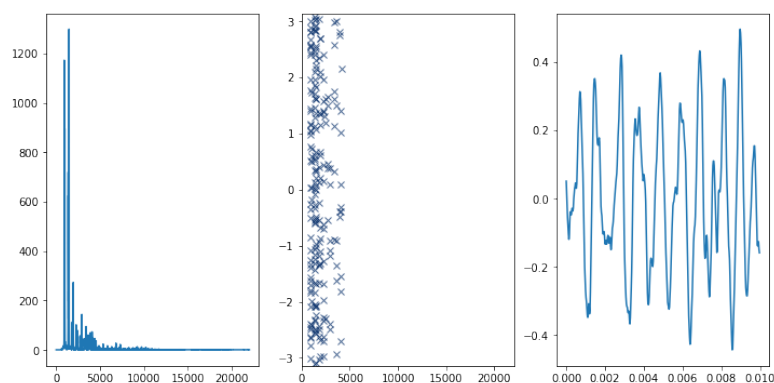


Рис. 3.21: Визуализация при случайных углах

Для этого сегмента изменение фазовой структуры имеет слышимый эффект, особенно случайные углы.

Если ухо использует что-то вроде автокорреляции для анализа подобных звуков, мы можем ожидать увидеть изменения в функции автокорреляции при изменении фазовой структуры.

3.5 Построение АКФ

Следующие функции строят АКФ для этих сегментов:

```

1 def autocorr(segment):
2     corrs = np.correlate(segment.ys, segment.ys, mode='same')
3     N = len(corrs)
4     lengths = range(N, N//2, -1)
5
6     half = corrs[N//2:].copy()
7     half /= lengths
8     half /= half[0]
9     return half

```

Листинг 3.30: Функция autocorr

```

1 def plot_acf(spectrum):
2     corrs = autocorr(spectrum.make_wave())
3     thinkplot.plot(corrs[:200], linewidth=1)

```

Листинг 3.31: Функция plot_acf

```

1 plot_acf(spectrum)
2 plot_acf(spectrum2)
3 plot_acf(spectrum3)
4 plot_acf(spectrum4)
5 thinkplot.config(xlabel='Lag', ylabel='Correlation', ylim=[-1.05,
    1.05])

```

Листинг 3.32: Построение АКФ

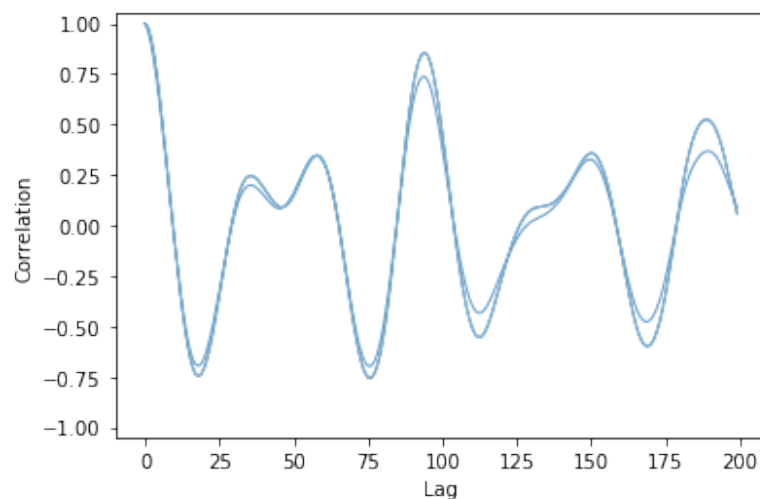


Рис. 3.22: Построение АКФ

Как мне кажется, изменение фазовой структуры оказывает некоторое влияние на АКФ, но не так сильно, и там нет ничего, что явно объясняет изменения в воспринимаемом звуке.

По крайней мере, для таких звуков, которые имеют простую гармоническую структуру, мы не слышим изменений в фазовой структуре, при условии, что гармоническая структура не изменилась.

Возможное исключение - звуки с низкой амплитудой на основной частоте. В этом случае мы используем что-то вроде автокорреляции для восприятия высоты звука, и это намек на то, что этот анализ может быть более чувствительным к фазовой структуре. Однако в АКФ нет ничего очевидного, объясняющего этот эффект.

Глава 4

Выводы

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