Лабораторная работа № 6. Дискретное косинусное преобразование.

3530901/80201, Шелаев Н. Р.

31 мая 2021 г.

Оглавление

Список иллюстраций

Листинги

Глава 1

Синтез

Сначала изучим синтез сигналов.

```
    from thinkdsp import CosSignal, SumSignal

    def synthesize1(amps, fs, ts):
      components = [CosSignal(freq, amp) for amp, freq in zip
    (amps, fs)]
      signal = SumSignal(*components)
      ys = signal.evaluate(ts)
      return ys

```

Листинг 1.1: Первая функция для синтеза

```
    def synthesize2(amps, fs, ts):
      args = np.outer(ts, fs)
      M = np.cos(PI2 * args)
      ys = np.dot(M, amps)
      return ys

```

Листинг 1.2: Вторая функция для синтеза

```
    ys1 = synthesize1(amps, fs, ts)
    ys2 = synthesize2(amps, fs, ts)
    np.max(np.abs(ys1 - ys2))

```

Листинг 1.3: Сравнение этих двух функций

Эти две функции выдают один и тот же результат.

Глава 2

Анализ

Теперь приступим к анализу сигналов.

```
    def analyze1(ys, fs, ts):
      args = np.outer(ts, fs)
      M = np.cos(PI2 * args)
      amps = np.linalg.solve(M, ys)
      return amps

```

Листинг 2.1: Первая функция для анализа

```
    def test1():
      N = 4.0
      time_unit = 0.001
      ts = np.arange(N) / N * time_unit
      max_freq = N / time_unit / 2
      fs = np.arange(N) / N * max_freq
      args = np.outer(ts, fs)
      M = np.cos(PI2 * args)
      return M

    M = test1()
    M.transpose().dot(M)

```

Листинг 2.2: Первая функция для тестирования

```
    def test2():
      N = 4.0
      ts = (0.5 + np.arange(N)) / N
      fs = (0.5 + np.arange(N)) / 2
      args = np.outer(ts, fs)
      M = np.cos(PI2 * args)
      return M

```

```
    M = test2()
    M.transpose().dot(M)

```

Листинг 2.3: Вторая функция для тестирования

С помощью двух тестовых функций мы посмотрели работу первой функции для анализа и пришли к выводу, что можно использовать матричное умножение. Это сильно ускорит работу функции.

```
    def analyze2(ys, fs, ts):
      args = np.outer(ts, fs)
      M = np.cos(PI2 * args)
      amps = M.dot(ys) / 2
      return amps

```

Листинг 2.4: Вторая функция для анализа

Глава 3

Функция ДКП-IV

Что же это за функция?

```
    def dct_iv(ys, *args):
      N = len(ys)
      ts = (0.5 + np.arange(N)) / N
      fs = (0.5 + np.arange(N)) / 2
      args = np.outer(ts, fs)
      M = np.cos(PI2 * args)
      amps = np.dot(M, ys) / 2
      return amps

    def inverse_dct_iv(amps): return dct_iv(amps) * 2

```

Листинг 3.1: Функция ДКП-IV

Убедились, что эта функция работает правильно.

Глава 4

Класс Dct

Теперь изучим предоставленный нам класс `Dct`.

```
from thinkdsp import TriangleSignal

signal = TriangleSignal(freq = 400)
wave = signal.make_wave(duration=1.0, framerate=10000)
dct = wave.make_dct()
dct.plot()

```

Листинг 4.1: Применение этого класса

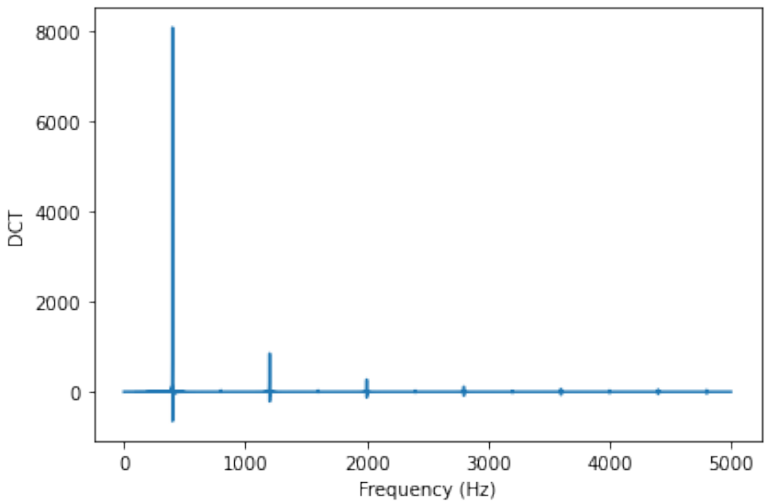

Рис. 4.1: Результат работы функции для тругольного сигнала

```
signal = TriangleSignal(freq = 400, offset = 0)
wave = signal.make_wave(duration=1.0, framerate=10000)
```

```
3       wave.ys *= -1
4       wave.make_dct().plot()
5
```

Листинг 4.2: Обратный Dct

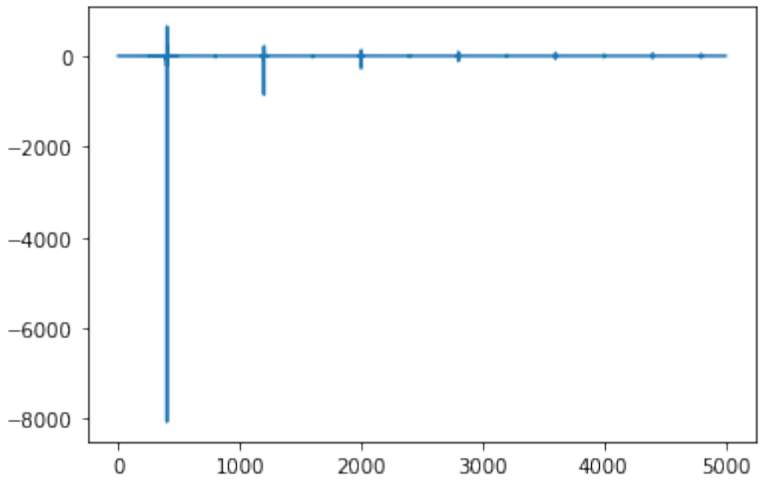

Рис. 4.2: Применение обратного Dct

Глава 5

Упражнения

5.1 Задание 1

Исследование скорости работы полученных функций.

```
    import scipy.fftpack
    from scipy.stats import linregress

    signal = UncorrelatedGaussianNoise()
    noise = signal.make_wave(duration=1.0, framerate=16384)

    def scipy_dct(ys, freqs, ts):
      return scipy.fftpack.dct(ys, type = 5)

    def plot_bests(ns, bests):
      plt.plot(ns, bests)
      x = np.log(ns)
      y = np.log(bests)
      t = linregress(x,y)
      slope = t[0]
      return slope

    def run_speed_test(ns, func):
      results = []
      for N in ns:
        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

```

Листинг 5.1: Подготовка к исследованию

```
    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)
    bests4 = run_speed_test(ns, dct_iv)
    plt.plot(ns, bests, label = 'analyze1')
    plt.plot(ns, bests2, label = 'analyze2')
    plt.plot(ns, bests3, label = 'fftpack.dct')
    plt.plot(ns, bests4, label = 'dct_iv')

```

Листинг 5.2: Кто быстрее?

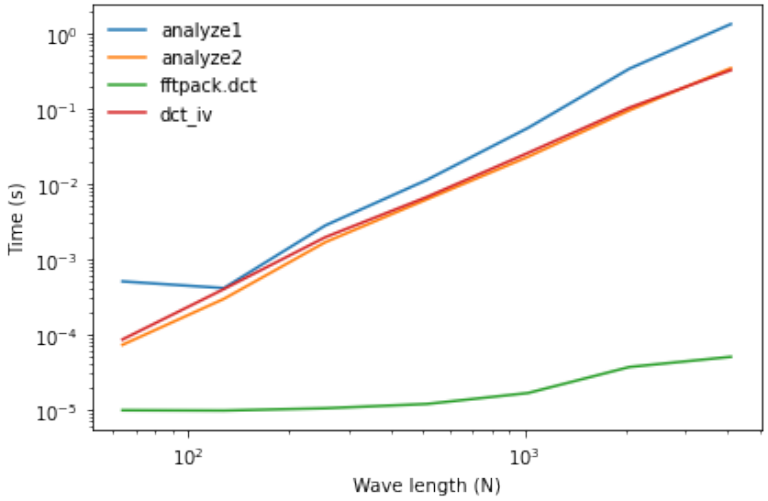

Рис. 5.1: Победитель очевиден

5.2 Задание 2

Сжатие звука с помощью функции ДКП.

```
    from thinkdsp import read_wave

    wave = read_wave('100475__iluppai__saxophone-weep.wav')
    segment = wave.segment(start = 1.2, duration = 0.5)
    seg_dct = segment.make_dct()
    seg_dct.plot(high = 4000)

```

Листинг 5.3: Получение исходного сигнала

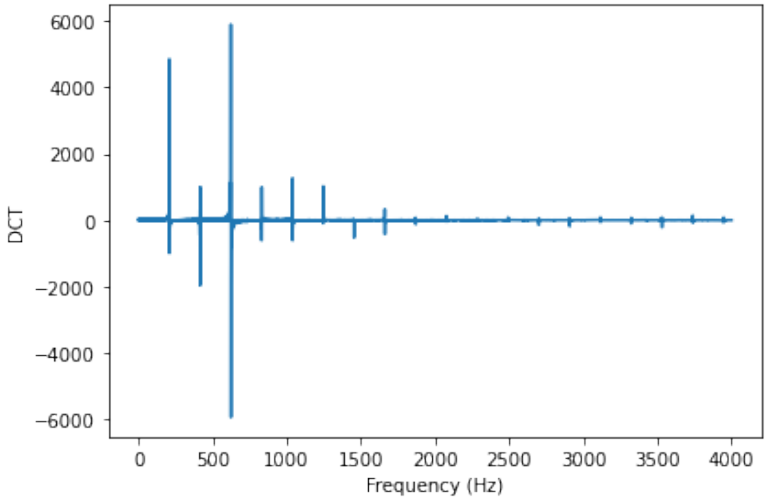

Рис. 5.2: До сжатия сигнала

```
    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)

```

Листинг 5.4: Функция для сжатия сигнала

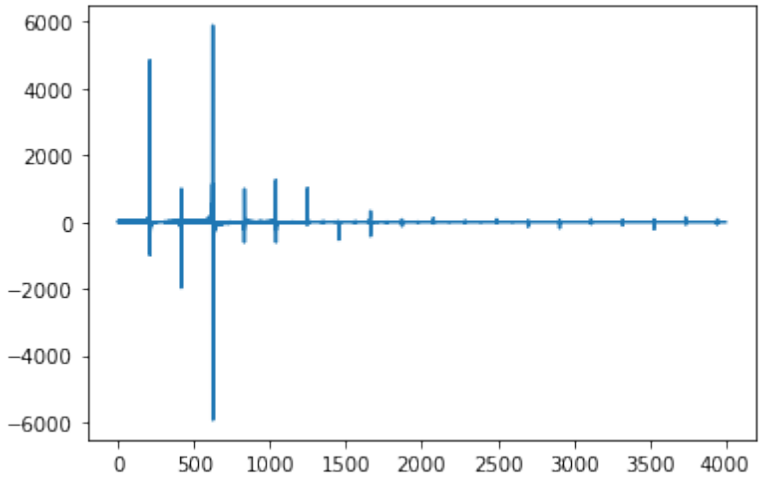

Рис. 5.3: После сжатия

```
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)

```

Листинг 5.5: Применение функции для сжатия сигнала

Для большинства сегментов сигнала доля сжатия составила примерно 75-85%. При этом звучание сигнала почти не изменилось.

5.3 Задание 3

Исследуем код в файле `phase.ipynb`.

```
from thinkdsp import SawtoothSignal

signal = SawtoothSignal(freq = 500, offset = 0)
wave = signal.make_wave(duration=0.5, framerate=40000)
wave.segment(duration = 0.01).plot()
spectrum = wave.make_spectrum()
spectrum.plot()

```

Листинг 5.6: Начало исследования

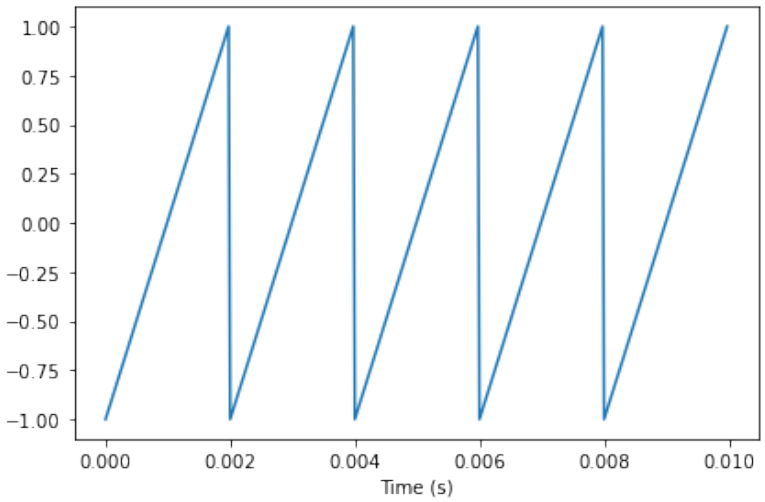

Рис. 5.4: Сегмент пилообразного сигнала

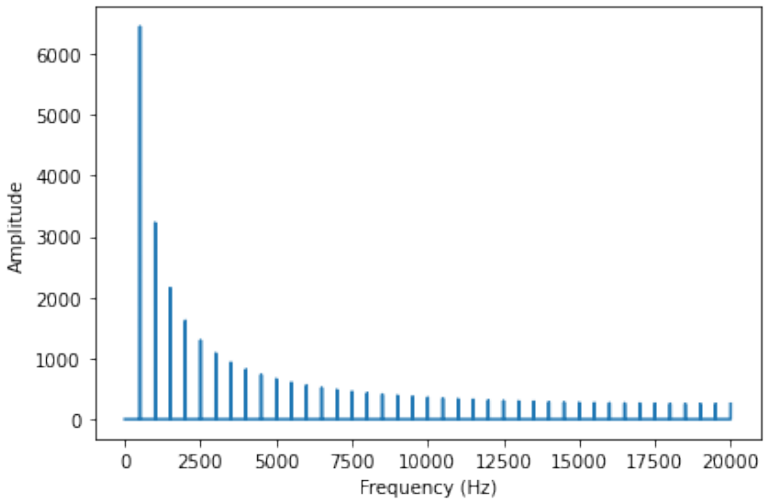

Рис. 5.5: Спектр пилообразного сигнала

```
1     def plot_angle(spectrum, thresh = 1):
2       angles = spectrum.angles
3       angles[spectrum.amps < thresh] = np.nan
4       plt.plot(spectrum.fs, angles, 'x')
5       decorate(xlabel = 'Frequency (Hz)', ylabel = 'Phase (
    radian)')
6
7     plot_angle(spectrum, thresh = 0)
8     plot_angle(spectrum, thresh = 1)
9
```

Листинг 5.7: Угловая часть спектра

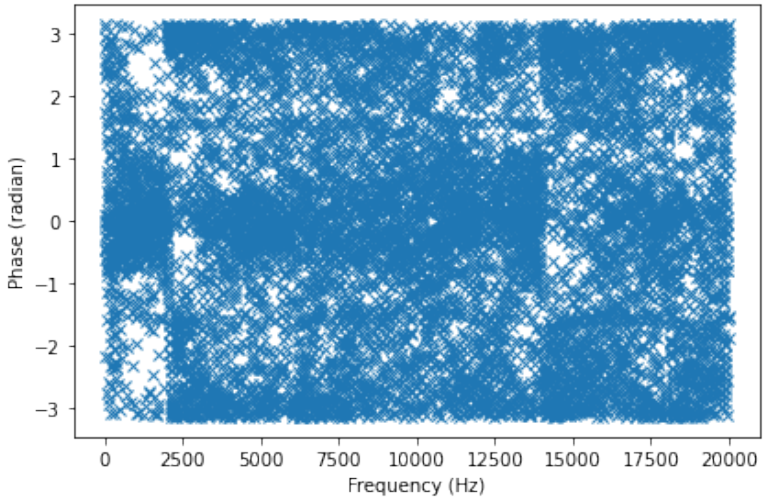

Рис. 5.6: Все углы спектра

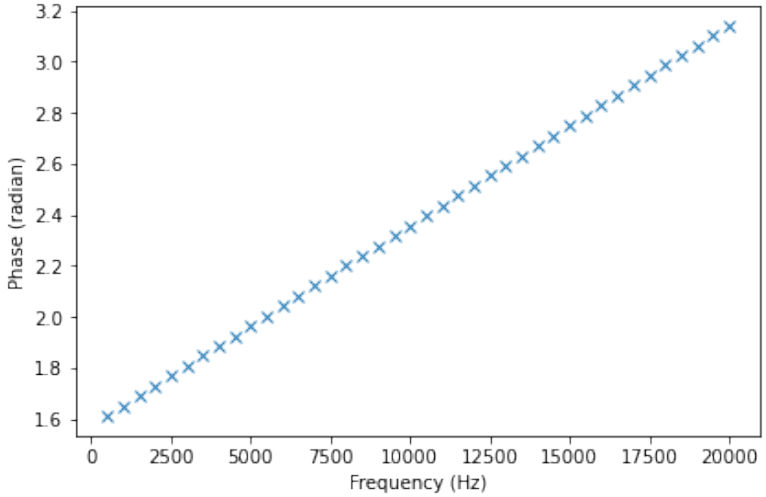

Рис. 5.7: Структура углов

```
def plot_three(spectrum, thresh = 1):
  plt.figure(figsize = (10, 4))
  plt.subplot(1, 3, 1)
  spectrum.plot()
  plt.subplot(1, 3, 2)
  plot_angle(spectrum, thresh = thresh)
  plt.subplot(1, 3, 3)
  wave = spectrum.make_wave()
  wave.unbias()
```

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

```

Листинг 5.8: Функция для 3-х графиков

```
def zero_angle(spectrum):
    res = spectrum.copy()
    res.hs = res.amps
    return res

spectrum2 = zero_angle(spectrum)
plot_three(spectrum2)

```

Листинг 5.9: Все углы равны нулю

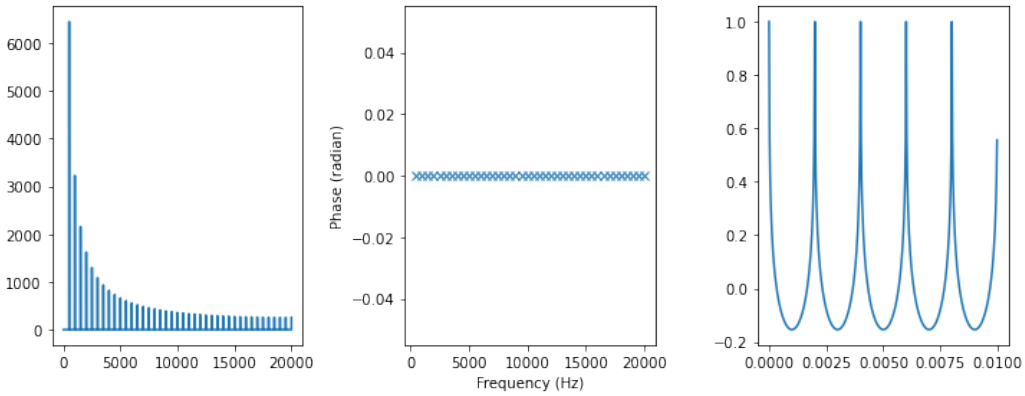

Рис. 5.8: Нулевые углы

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

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

```

Листинг 5.10: Функция для поворота углов

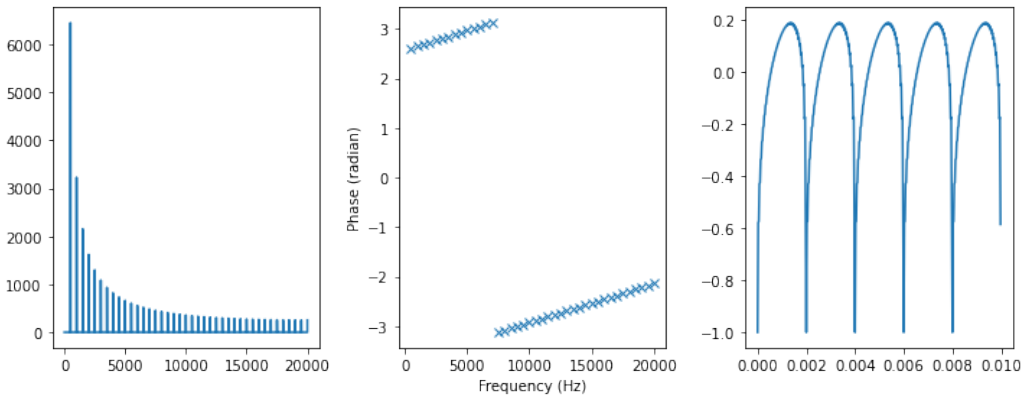

Рис. 5.9: Результат поворота углов

```
def random_angle(spectrum):
  res = spectrum.copy()
  angles = np.random.uniform(0, PI2, len(spectrum))
  res.hs *= np.exp(1j * angles)
  return res

spectrum4 = random_angle(spectrum)
plot_three(spectrum4)

```

Листинг 5.11: Функция для генерации случайных углов

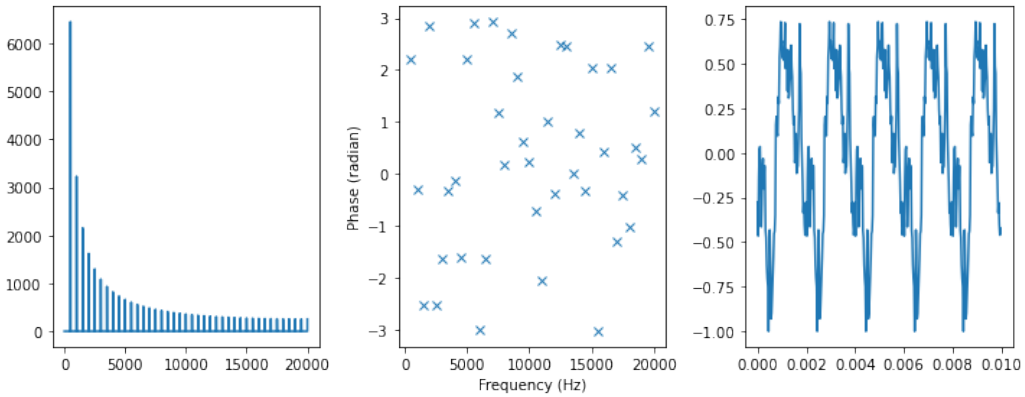

Рис. 5.10: Слачайные углы

```
from thinkdsp import read_wave

wave = read_wave('120994__thirsk__120-oboe.wav')
segment = wave.segment(start = 0.05, duration = 0.9)
```

```
spectrum = segment.make_spectrum()
plot_three(spectrum, thresh = 50)

```

Листинг 5.12: Исходный сигнал

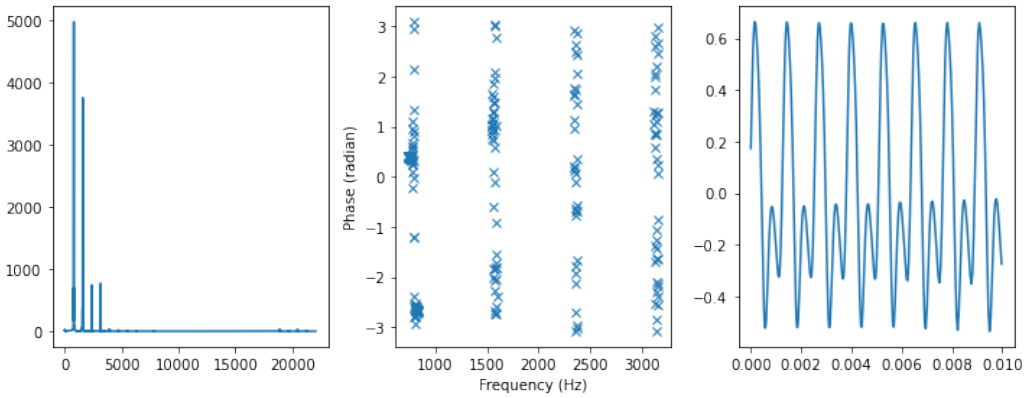

Рис. 5.11: Исходный сегмент

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

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

spectrum4 = random_angle(spectrum)
plot_three(spectrum4, thresh = 50)

```

Листинг 5.13: Применение функций

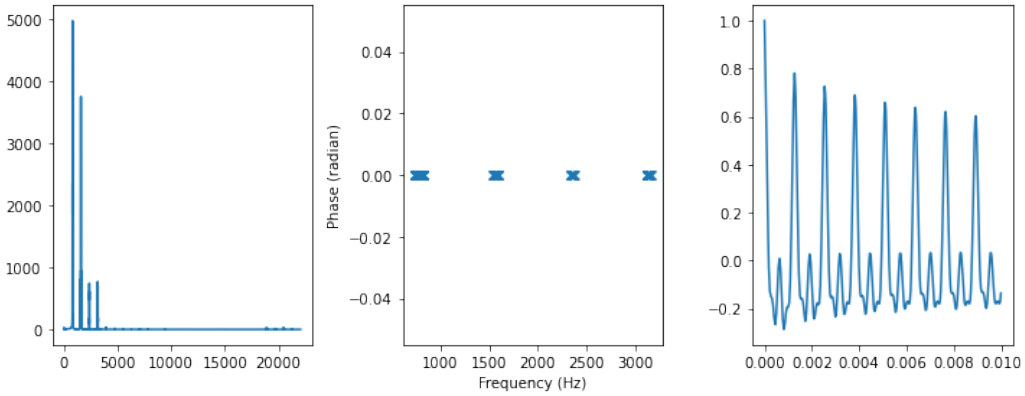

Рис. 5.12: Занулили углы

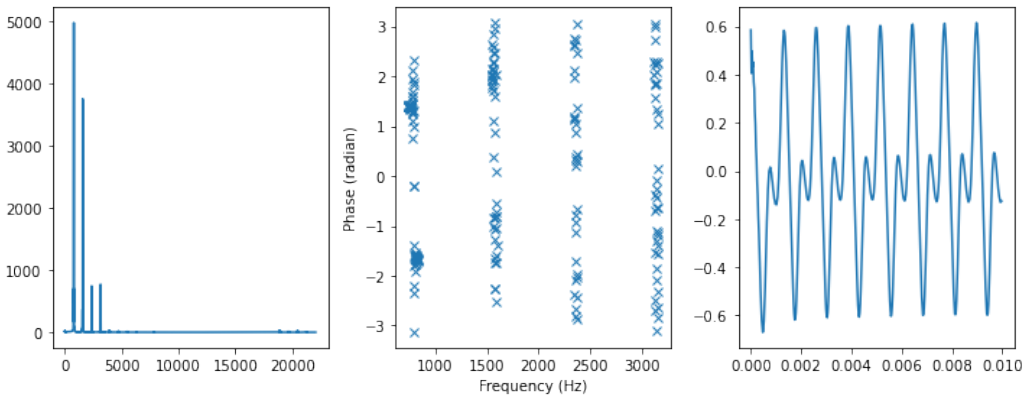

Рис. 5.13: Повернули

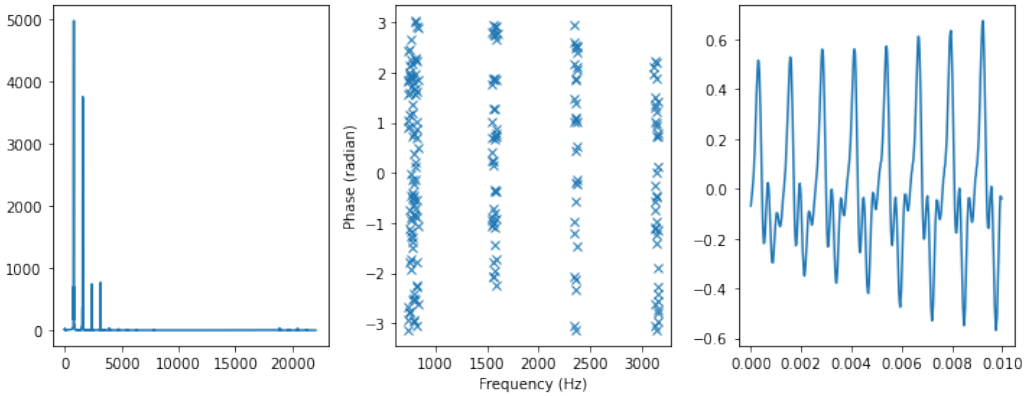

Рис. 5.14: Случайные значения

```
wave = read_wave('100475__iluppai__saxophone-weep.wav')
wave.make_audio()
segment = wave.segment(start = 1.9, duration = 0.6)
spectrum = segment.make_spectrum()
plot_three(spectrum, thresh = 50)

```

Листинг 5.14: Возьмём другой сигнал

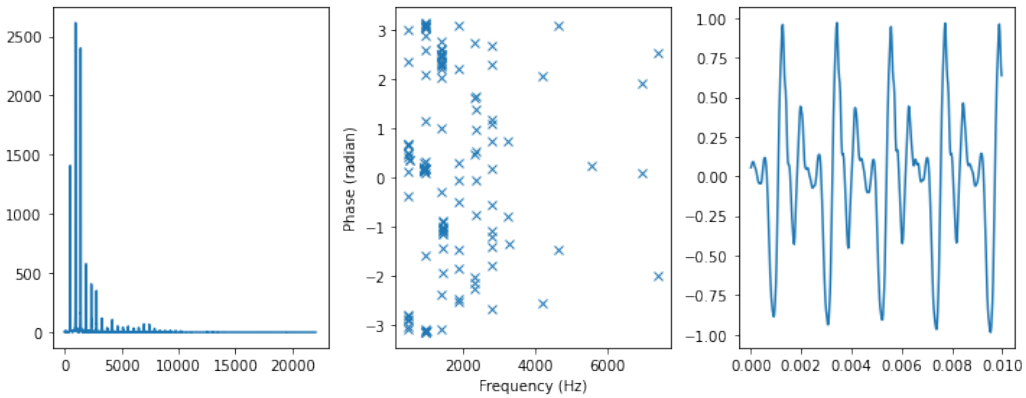

Рис. 5.15: Исходный сигнал

```
spectrum2 = zero_angle(spectrum)
plot_three(spectrum2, thresh = 50)
...

```

Листинг 5.15: Применение функций для другого сигнала

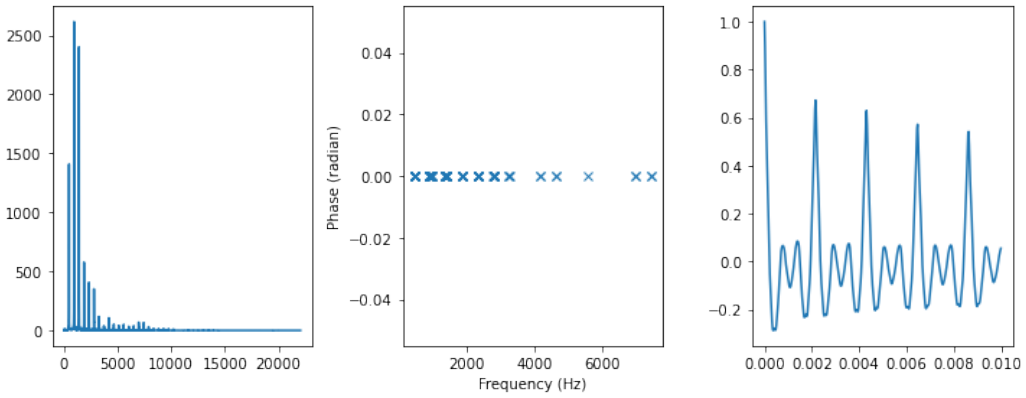

Рис. 5.16: Углы = 0

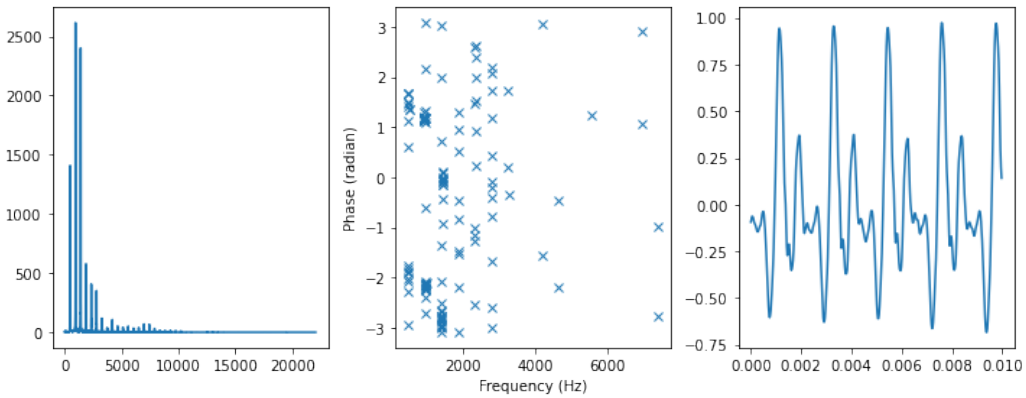

Рис. 5.17: Поворот углов на 1 радиан

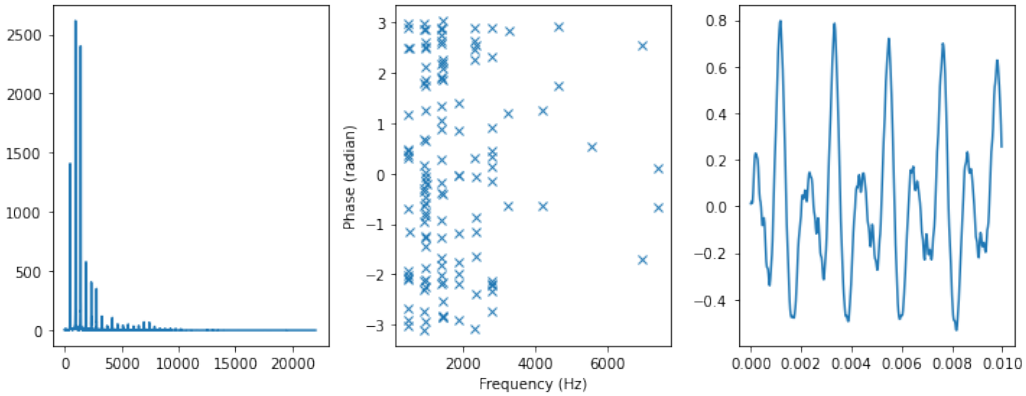

Рис. 5.18: Random

```
spectrum.high_pass(600)
spectrum.plot(high = 4000)
plot_three(spectrum2, thresh = 50)

```

Листинг 5.16: Применили ФВЧ для сигнала

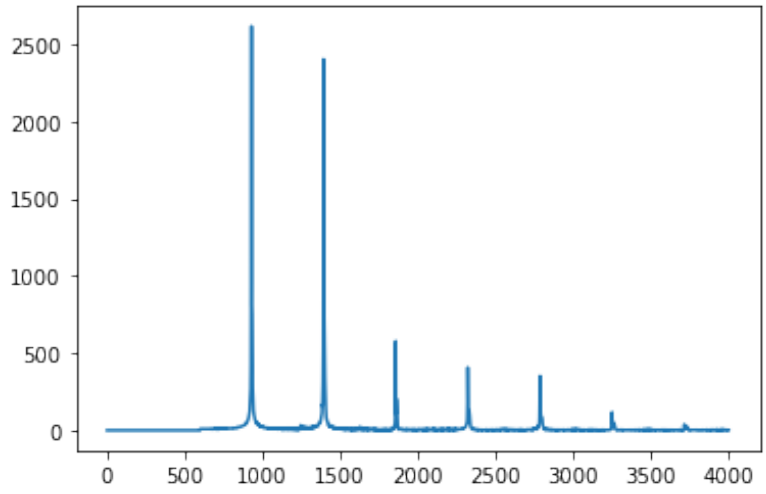

Рис. 5.19: Спектр сигнала изменился

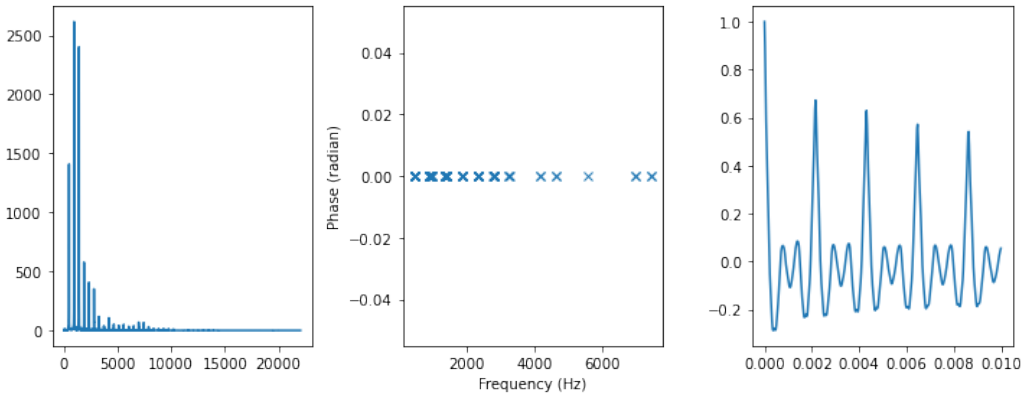

Рис. 5.20: Исходный сигнал после ФВЧ

```
    spectrum2 = zero_angle(spectrum)
    plot_three(spectrum2, thresh = 50)
    ...

```

Листинг 5.17: Применение функций сигнала после ФВЧ

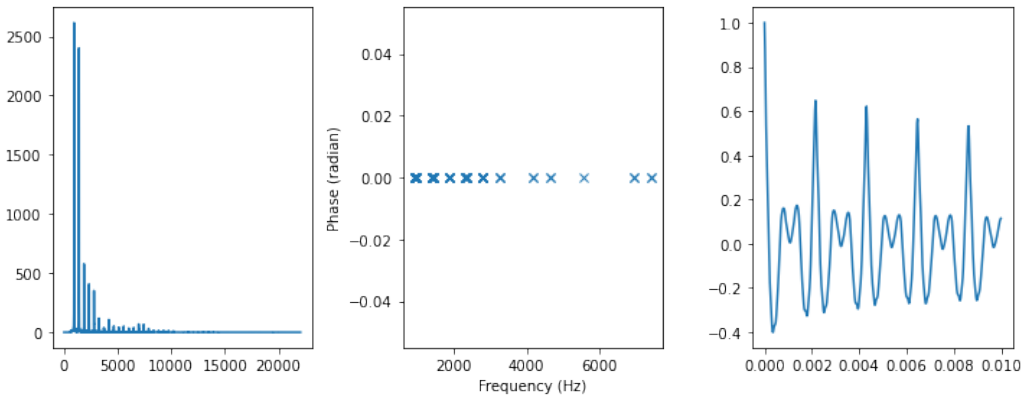

Рис. 5.21: Обнуление углов

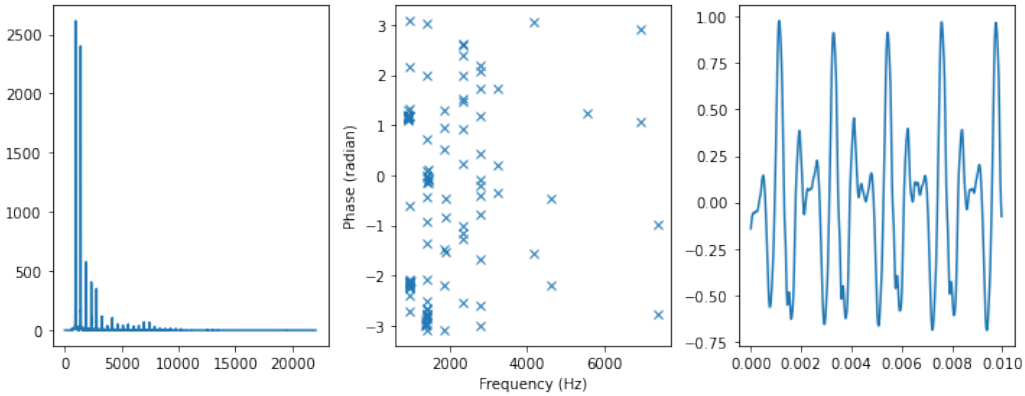

Рис. 5.22: 1 радиан

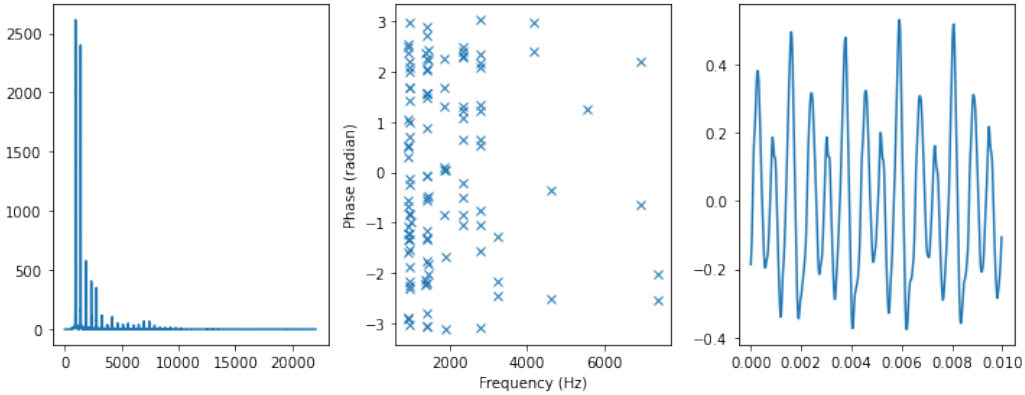

Рис. 5.23: Случайность

Мы не услышали каких-либо изменений в сигналах.

Глава 6

Вывод

В данной работе мы изучили и применили различные функции для анализа и синтеза сигналов. Встроенная функция `Dct` для анализа оказалась быстрее других алгоритмов.Также применили функции ДКП для сжатия сигналов.