Лабораторная работа № 7. Дискретное преобразование Фурье.

3530901/80201, Шелаев Н. Р.

31 мая 2021 г.

Оглавление

Список иллюстраций

Листинги

Глава 1

Комплексные сигналы

Попробуем построить какой-нибудь комплексный сигнал.

```
1       class ComplexSinusoid(Sinusoid):
2
3       def evaluate(self, ts):
4           phases = PI2 * self.freq * ts + self.offset
5           ys = self.amp * np.exp(1j * phases)
6           return ys
7
8       signal = ComplexSinusoid(freq=1, amp=0.6, offset=1)
9       wave = signal.make_wave(duration = 1, framerate = 4)
10
```

Листинг 1.1: Строим комплексную синусоиду

```
1       from thinkdsp import SumSignal
2
3       def synthesize1(amps, freqs, ts):
4           components = [ComplexSinusoid(freq, amp) for amp,
    freq in zip(amps, freqs)]
5           signal = SumSignal(*components)
6           ys = signal.evaluate(ts)
7           return ys
8
9       amps = np.array([0.6, 0.25, 0.1, 0.05])
10      freqs = [100, 200, 300, 400]
11      framerate = 11025
12      ts = np.linspace(0, 1, framerate, endpoint = False)
13      ys = synthesize1(amps, freqs, ts)
14      n = 500
15      plt.plot(ts[:n], ys[:n].real)
16      plt.plot(ts[:n], ys[:n].imag)
17
```

Листинг 1.2: Исследуем полученный сигнал

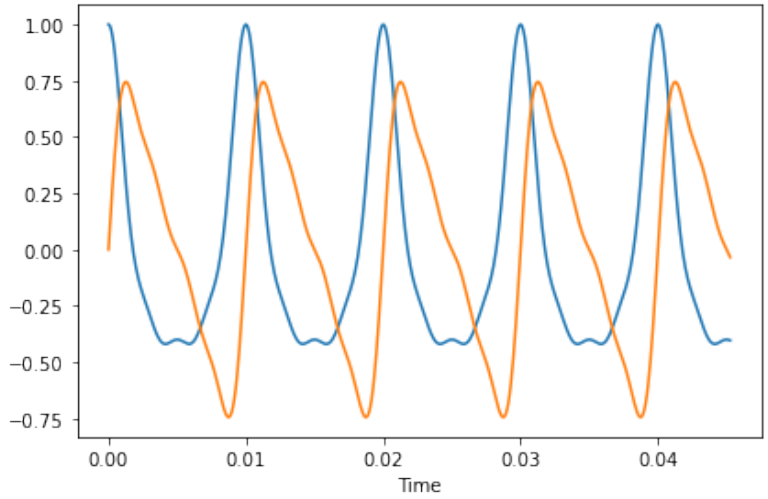

Рис. 1.1: Вещественная и мнимая части комплексной синусоиды

Действительная часть - смесь косинусов, а мнимая часть - смесь синусов. Они содержат одни и те же частотные компоненты с одинаковыми амплитудами, поэтому для нас они звучат одинаково.

```
    def synthesize2(amps, freqs, ts):
        args = np.outer(ts, freqs)
        M = np.exp(1j * PI2 * args)
        ys = np.dot(M, amps)
        return ys

    amps = np.array([0.6, 0.25, 0.1, 0.05])
    ys = synthesize2(amps, freqs, ts)
    wave = Wave(ys.real, framerate)
    phi = 1.5
    amps2 = amps * np.exp(1j * phi)
    ys2 = synthesize2(amps2, freqs, ts)
    n = 500
    plt.plot(ts[:n], ys.real[:n], label=r'$\phi_0 = 0$')
    plt.plot(ts[:n], ys2.real[:n], label=r'$\phi_0 = 1.5$')

```

Листинг 1.3: Применим функцию с матричным умножением

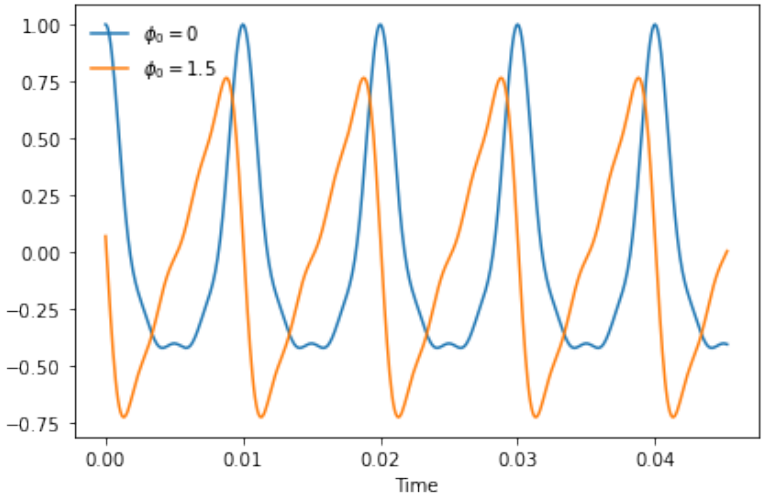

Рис. 1.2: Повернули амплитуду на 1,5 радиана

Поворот компонентов с одиновым смещением фазы изменил форму сигнала. Компоненты имеют разные периоды, и одно и то же смещение оказывает различное влияние на каждый компонент.

Глава 2

Анализ

Попробуем вычислить комплексные амплитуды компонент, зная частоты сигнала.

```
1       def analyze1(ys, freqs, ts):
2          args = np.outer(ts, freqs)
3          M = np.exp(1j * PI2 * args)
4          amps = np.linalg.solve(M, ys)
5          return amps
6
7       n = len(freqs)
8       amps2 = analyze1(ys[:n], freqs, ts[:n])
9       N = 4
10      ts = np.arange(N) / N
11      freqs = np.arange(N)
12      args = np.outer(ts, freqs)
13      M = np.exp(1j * PI2 * args)
14      MstarM = M.conj().transpose().dot(M)
15
```

Листинг 2.1: Применение первой функции для анализа

Получилась унитарная матрица с точностью до дополнительного фактора `N`. Значит, ответ правильный.

```
1       def analyze2(ys, freqs, ts):
2          args = np.outer(ts, freqs)
3          M = np.exp(1j * PI2 * args)
4          amps = M.conj().transpose().dot(ys) / N
5          return amps
6
7       N = 4
8       amps = np.array([0.6, 0.25, 0.1, 0.05])
9       freqs = np.arange(N)
10      ts = np.arange(N) / N
11      ys = synthesize2(amps, freqs, ts)
```

```
12        amps3 = analyze2(ys, freqs, ts)
13
```

Листинг 2.2: Применение второй функции для анализа

Снова получили правильный результат с точностью до ошибок округления.

```
1         def synthesis_matrix(N):
2             ts = np.arange(N) / N
3             freqs = np.arange(N)
4             args = np.outer(ts, freqs)
5             M = np.exp(1j * PI2 * args)
6             return M
7
8         def dft(ys):
9             N = len(ys)
10            M = synthesis_matrix(N)
11            amps = M.conj().transpose().dot(ys)
12            return amps
13
```

Листинг 2.3: Улучшение функции

Полученный результат совпал с ответом функции `np.fft.fft` в пределах ошибок округления.

Глава 3

Реальные сигналы

Протестируем полученную функцию DFT на реальных сигналах.

```
1       from thinkdsp import SawtoothSignal
2
3       framerate = 10000
4       signal = SawtoothSignal(freq = 500)
5       wave = signal.make_wave(duration=0.1, framerate=
    framerate)
6       hs = dft(wave.ys)
7       amps = np.abs(hs)
8       plt.plot(amps)
9
```

Листинг 3.1: Тест на реальном сигнале

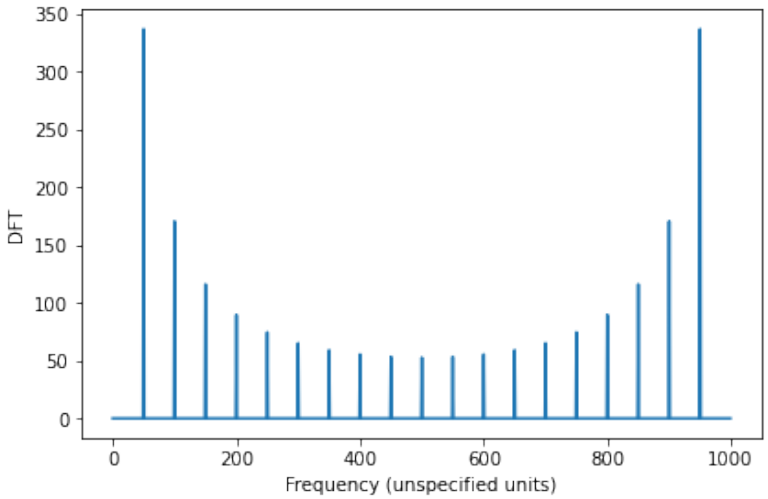

Рис. 3.1: Результат

```
    N = len(hs)
    fs = np.arange(N) * framerate / N
    plt.plot(fs[:(N // 2 + 1)], amps[:(N //2 + 1)])

```

Листинг 3.2: Берём левую половину сигнала

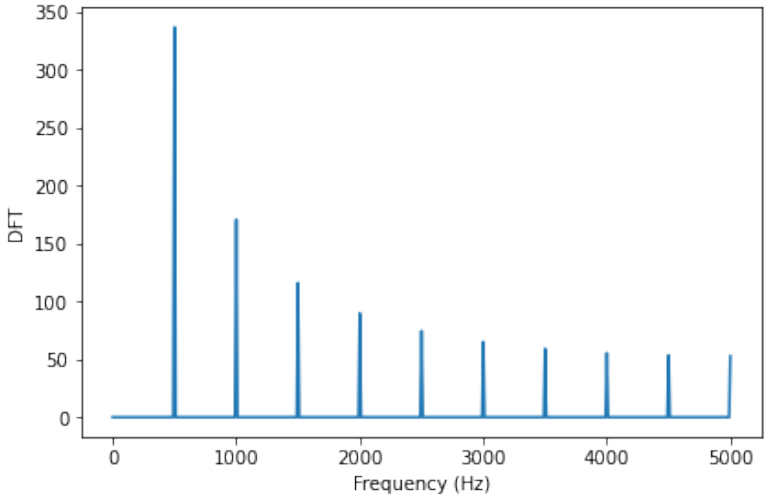

Рис. 3.2: Первая половина сигнала

```
    from thinkdsp import TriangleSignal

    wave = TriangleSignal(freq = 1).make_wave(duration = 1,
 framerate = 8)
    wave2 = ComplexSinusoid(freq = 1).make_wave(duration =
1, framerate = 8)
    plt.plot(wave2.ts, wave2.ys.real)
    plt.plot(wave2.ts, wave2.ys.imag)

    hs = np.fft.fft(wave.ys)
    plt.plot(abs(hs))

    hs = np.fft.fft(wave2.ys)
    plt.plot(abs(hs))

```

Листинг 3.3: Сравнение реального сигнала с комплексным

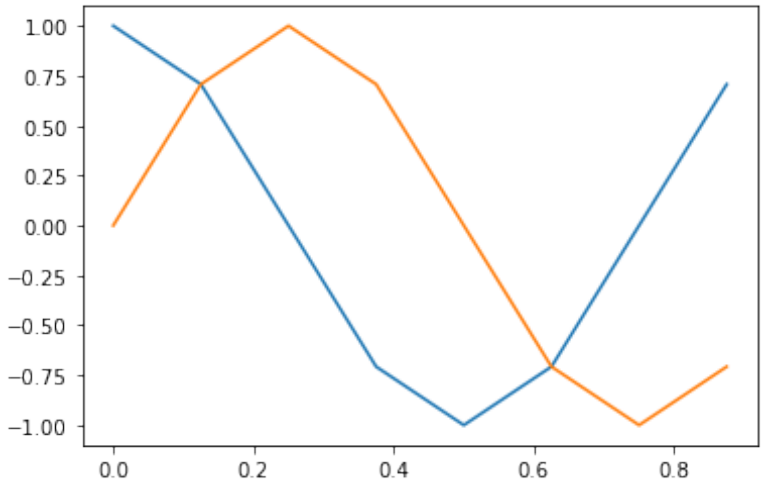

Рис. 3.3: Комплексная синусоида

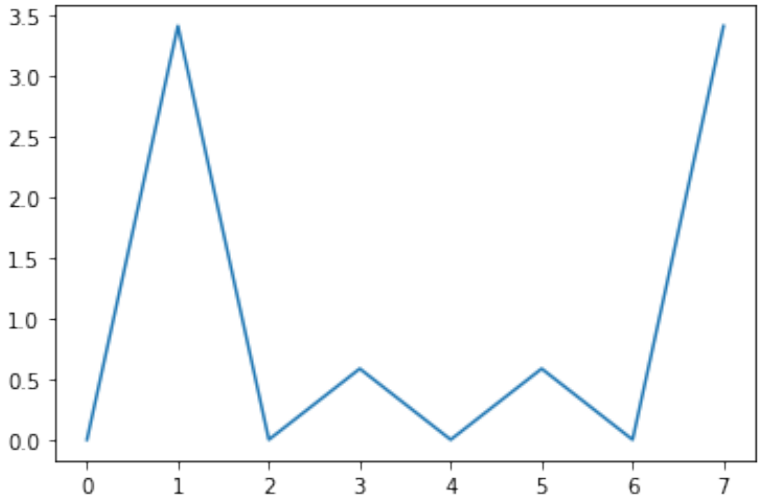

Рис. 3.4: FFT реального сигнала - симметричный

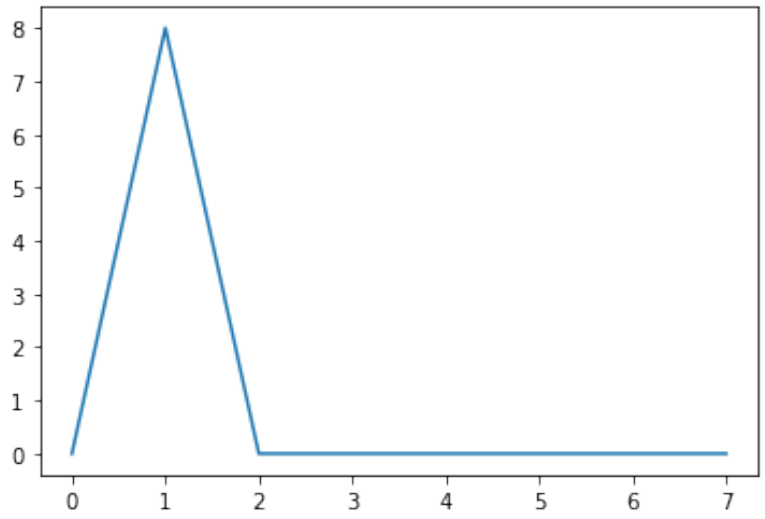

Рис. 3.5: FFT комплексного сигнала - несимметричный

Глава 4

Упражнения

4.1 Задание 2

Рекурсивный алгоритм для `DFT` по лемме Дэниелсона-Ланцоша.

$$DFT(y)[n] = DFT(e)[n] + e^{-\frac{2\pi in}{N}} \cdot DFT(o)[n] \tag{4.1}$$

где DFT(y)[n] - n-ый элемент DFT от y,
e - массив сигнала, содержащий только четные элементы y
o - только нечетные элементы y.

```
1       def dft(ys):
2           N = len(ys)
3           ts = np.arange(N) / N
4           freqs = np.arange(N)
5           args = np.outer(ts, freqs)
6           M = np.exp(1j * PI2 * args)
7           amps = M.conj().transpose().dot(ys)
8           return amps
9
```

Листинг 4.1: Функция DFT

Сравнили результат её работы с ответом функции `np.fft.fft`. Результаты совпали.

```
1       def fft_norec(ys):
2           N = len(ys)
3           He = np.fft.fft(ys[::2])
4           Ho = np.fft.fft(ys[1::2])
5           ns = np.arange(N)
6           W = np.exp(-1j * PI2 * ns / N)
7           return (np.tile(He, 2) + W * np.tile(Ho, 2))
```

Листинг 4.2: Функция для разделения исходного массива пополам

Проверили её работу - всё правильно.

```
    def fft(ys):
        N = len(ys)
        if N == 1: return ys
        He = fft(ys[::2])
        Ho = fft(ys[1::2])
        ns = np.arange(N)
        W = np.exp(-1j * PI2 * ns / N)
        return (np.tile(He, 2) + W * np.tile(Ho, 2))
```

Листинг 4.3: Итоговая функция

Также получили правильные результаты. Эта реализация алгоритма работает за $n \log n$ и занимает $n \log n$ памяти. Возможно, его можно улучшить.

Глава 5

Вывод

В данной работе мы изучили дискретное преобразование Фурье и реализовали алгоритмы для его использования с различными сигналами. В последнем упражнении нам удалось немного улучшить полученный алгоритм.