Санкт-Петербургский государственный политехнический
университет Петра Великого

**Высшая школа интеллектуальных систем и
суперкомпьютерных технологий**

Лабораторная работа

Дискретное преобразование Фурье

Работу выполнил студент
3-го курса, группа 3530901/80201
Сахибгареев Рамис Ринатович

Преподаватель:
Богач Наталья Владимировна

Санкт-Петербург 2021

Contents

List of Figures

Listings

1 Part 1: Research and execution of chap07

In this part we need to research and execute existing chap07.ipynb file, that contains information about Complex Signals, DFT and real and imaginary transformations.

File was successfully executed. Complex signal - is a signal, that have real and imaginary parts. They have same frequencies and amplitudes, but different phase (cos and sin signals), so human cannot recognize any difference between them. Also we can rotate the signal, but because different frequencies has different cycle time, signal changes too.

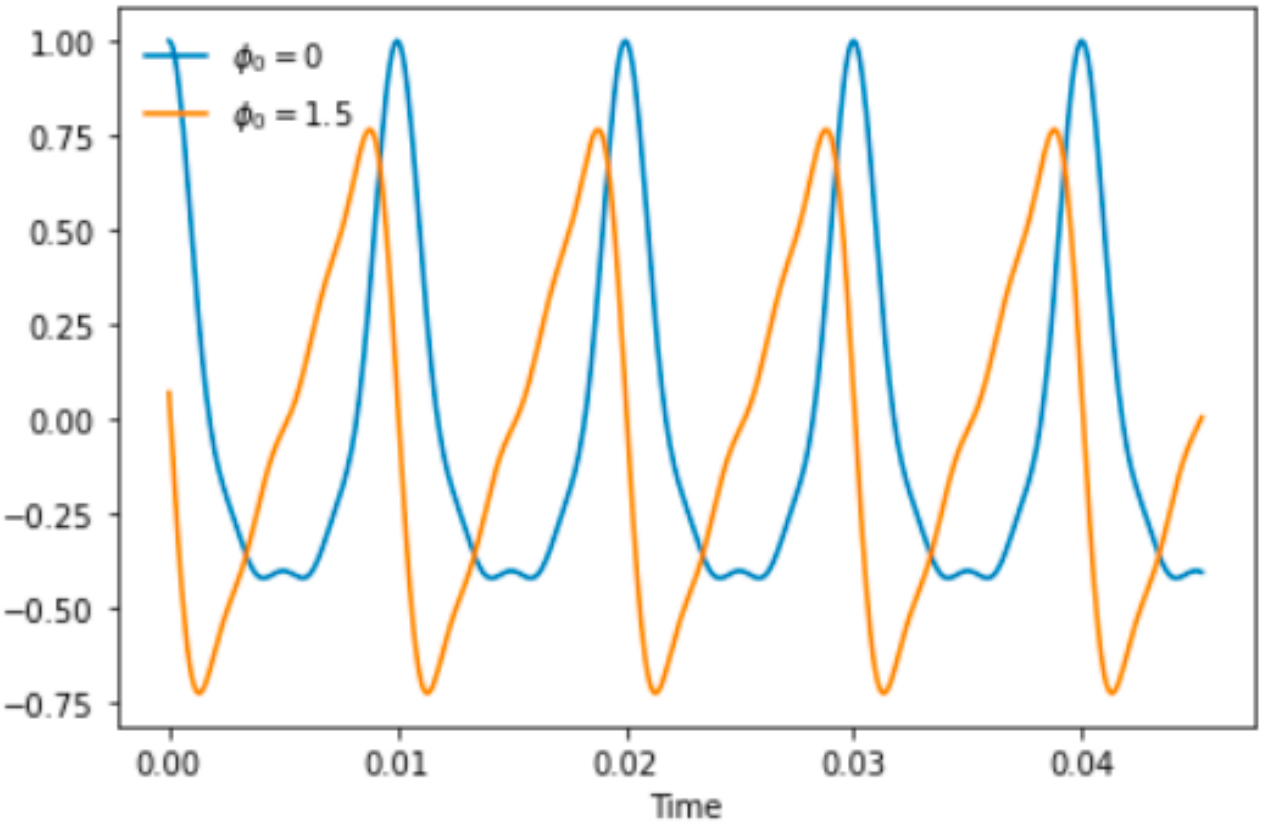

Figure 1: Phase changing result. Signals are different

We can also see, that full DFT also has negative frequencies.

```
In [51]: framerate = 10000
         signal = SawtoothSignal(freq=500)
         wave = signal.make_wave(duration=0.1, framerate=framerate)

In [52]: spectrum = wave.make_spectrum(full=True)

In [53]: spectrum.plot()
         decorate(xlabel='Frequency (Hz)', ylabel='DFT')
```

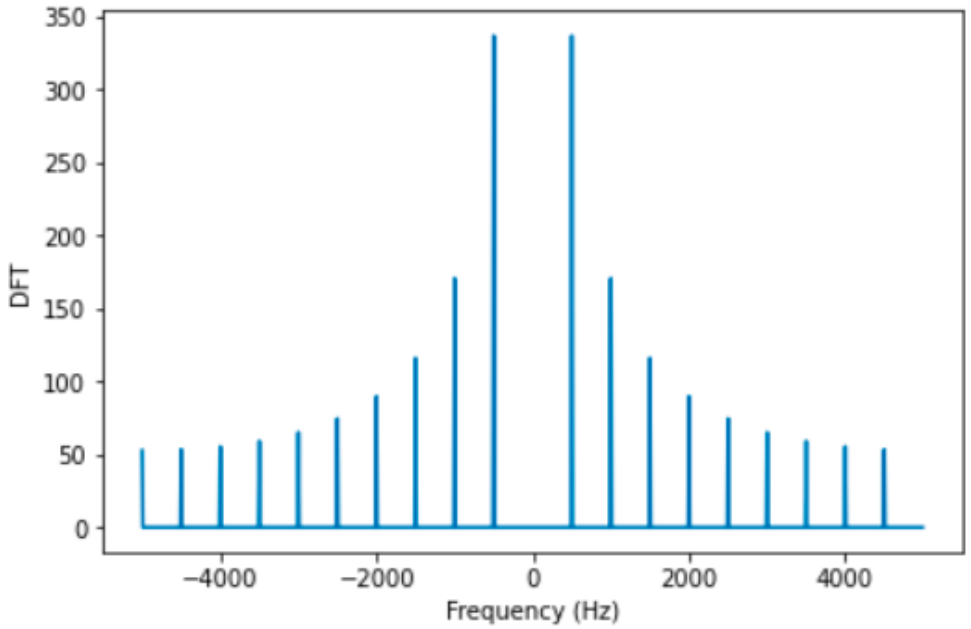

Figure 2: Full DFT result on a sawtooth signal signal

2 Part 2: Fast Fourier Transform (FFT) implementation

In this part we need to implement FFT using Damielson-Lancouz lemma. (o) and (e) is arrays, containing odd and even elements of ys correspondingly.

$$DFT(y)[n] = DFT(e)[n] + exp(-2 * \pi * i * n/N) * DFT(o)[n]$$

Standard DFT function take

$$N^2$$

time to be computed, which is good for a small N, but can take enormous time for big N. That's why we need to use recursive FFT, which takes around

$$N * logN$$

steps to be computed. When size of iteration will be small, we can use standard DFT, to get needed values.

To implement FFT let's use code of DFT, given in the previous part.

```
    def synthesis_matrix(N):
        ts = np.arange(N) / N
        freqs = np.arange(N)
        args = np.outer(ts, freqs)
        M = np.exp(1j * PI2 * args)
        return M
    def dft(ys):
        N = len(ys)
        M = synthesis_matrix(N)
        amps = M.conj().transpose().dot(ys)
        return amps

```

Listing 1: DFT definition

Using it and Damielson-Lancouz lemma let's implement Fast Fourier Transform function.

```
    def fft(ys):
        N = len(ys)
        for (i)
        if N <= 4:
            return dft(ys)

        even = fft(ys[::2])
        odd = fft(ys[1::2])

        ns = np.arange(N)
```

```
11
12          return np.tile(even, 2) + np.exp(-1j * PI2 * ns / N) * np.tile(odd, 2)
13
```

Listing 2: FFT definition

And we can compare its result to np.fft.ftt.

```
sig = SawtoothSignal(freq=1000)
ys = sig.make_wave(duration=0.0116).ys
np.sum(np.abs(np.fft.fft(ys) - fft(ys)))
```

```
3.0577363924977356e-13
```

Figure 3: FFT testing

We can see, that difference between those 2 functions is minor, so we can consider, that they are equals. However, we have other problem with out function: it works only of ys.len is power of 2 because of recursive division by 2. Otherwise even and odd arrays will be different sizes, so no broadcast operation will be possible.

3 Conclusion

We've learned, what is Complex Signals, how them can be transformed to the wave, how rotation affects them. Also we've learned, how DFT is performed and that is has full and real parts. We've created our own FFT function using Damielson-Lancouz lemma