

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
Солянкин Илья Андреевич

Преподаватель:
Богач Наталья Владимировна

Санкт-Петербург 2021

Содержание

1	Часть №1: Проверка шар07.іруnb	5
2	Часть №2: Использование ДПФ	6
3	Выводы	9

Список иллюстраций

1	Все запустилось	5
2	Получение БПФ	6
3	Результаты работы функции <code>dft</code>	7
4	Результаты работы функции <code>fft-norec</code>	7
5	Результаты работы функции <code>fft</code>	7

Листинги

1	Функция <code>dft</code>	6
2	Функция <code>fft-norec</code>	7
3	Функция <code>fft</code>	7

1 Часть №1: Проверка chap07.ipynb

В первом пункте седьмой лабораторной работы нам необходимо пройти по всем примерам из блокнота chap07.ipynb, запустив их и прочитав описания.

В данном блокноте сначала приводятся примеры работы со сложными синусоидами, также представлены примеры анализа сигнала, и примеры работы ДПФ с "real-valued" сигналом.

Все примеры успешно запустились, и для подтверждения этого я приложу скриншот успешного запуска последних строк документа:

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

```
[51] spectrum = wave.make_spectrum(full=True)
```

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

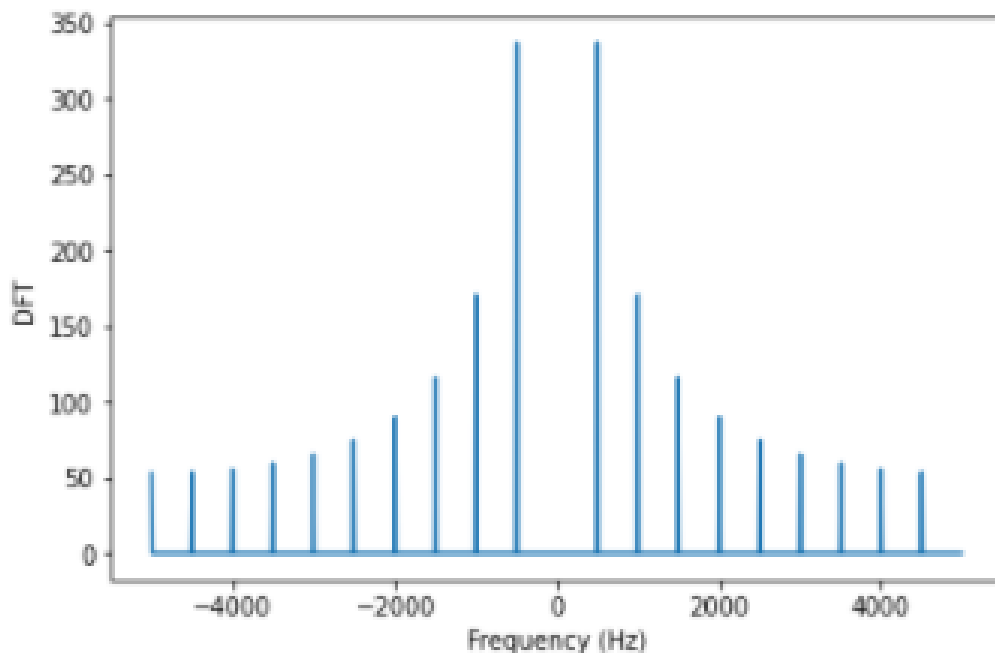


Рис. 1: Все запустилось

2 Часть №2: Использование ДПФ

Во втором пункте седьмой лабораторной работы было продемонстрировано использование ДПФ (дискретное преобразование Фурье) и обратное ДПФ в виде произведения матриц. Такие операции занимают время N^2 , где N - длина массива, что достаточно быстро для большинства применений, но есть более быстрый алгоритм: Быстрое Преобразование Фурье (БПФ) или FFT, занимающий $N \log(N)$

Ключевая вещь в БПФ это лемма Даниелсона-Ланкзоса, которая предлагает рекурсивный алгоритм для DFT:

1. Входящий массив y разделяется на четное число элементов e , и на нечётные элементы o .
2. Вычислить DFT e и o с помощью рекурсивных запросов.
3. Вычислить $DFT(y)$ для каждого значения n используя лемму Даниелсона-Ланкзоса.

В случае если длина исходного массива равна 1, $DFT(y) = y$. Или если длина y очень мала, можно вычислить её DFT с помощью матричного умножения, используя предварительно вычисленную матрицу.

Начнём с небольшого реального сигнала и вычислим его БПФ:

```
In [2]: ys = [-0.5, 0.1, 0.7, -0.1]
        hs = np.fft.fft(ys)
        print(hs)

[ 0.2+0.j -1.2-0.2j  0.2+0.j -1.2+0.2j]
```

Рис. 2: Получение БПФ

Теперь нам необходимо реализовать функцию `dft` для вычисления матрицы синтеза и сразу же протестируем ее:

```
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
```

Листинг 1: Функция `dft`

```
B [4]: hs2 = dft(ys)
np.sum(np.abs(hs - hs2))

Out[4]: 5.864775846765962e-16
```

Рис. 3: Результаты работы функции dft

После этого реализуем функцию `fft-norec`, которая будет разбивать входной массив и использовать `np.fft.fft` для вычисления БПФ половин:

```
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
6     ns = np.arange(N)
7     W = np.exp(-1j * PI2 * ns / N)
8
9     return np.tile(He, 2) + W * np.tile(Ho, 2)
```

Листинг 2: Функция `fft-norec`

```
B [6]: hs3 = fft_norec(ys)
np.sum(np.abs(hs - hs3))

Out[6]: 0.0
```

Рис. 4: Результаты работы функции `fft-norec`

И. наконец, реализуем функцию `fft`, которая похожа на предыдущую, но `np.fft.fft` заменена на рекурсию:

```
1 def fft(ys):
2     N = len(ys)
3     if N == 1:
4         return ys
5
6     He = fft(ys[:2])
7     Ho = fft(ys[1:2])
8
9     ns = np.arange(N)
10    W = np.exp(-1j * PI2 * ns / N)
11
12    return np.tile(He, 2) + W * np.tile(Ho, 2)
```

Листинг 3: Функция `fft`

```
B [8]: hs4 = fft(ys)
np.sum(np.abs(hs - hs4))

Out[8]: 1.6653345369377348e-16
```

Рис. 5: Результаты работы функции `fft`

В результате можно сказать, что полученная нами реализация БПФ занимает время, которое пропорционально $N \log(N)$ при создании и копировании массива, а также занимает аналогичное количество места.

3 Выводы

В результате выполнения данной лабораторной работы мы изучили, что такое ДПФ, БПФ. Мы создали функцию для вычисления БПФ, которая работает за $N \log(N)$ при создании и копировании массивов. Кроме того мы прошли по всем примерам из блокнота `chap07.ipynb`, запустив все блоки кода и прочитав всю информацию.