Санкт-Петербургский государственный политехнический
университет Петра Великого

**Высшая школа интеллектуальных систем и
суперкомпьютерных технологий**

Лабораторная работа

Дискретное преобразование Фурье

Выполнил студент гр. 3530901/80201 И.С. Иванов

Преподаватель: Н.В. Богач

Санкт-Петербург
2021

Содержание

Список иллюстраций

Листинги

1 Упражнение №1

В первом упражнении необходимо просмотреть все примеры из файла `chap07.ipyn` В этом файле приводятся примеры работы с сложными синусоидальными сигналами и примеры работы ДПФ.

Все примеры запустились и были изучены.

Complex sinusoid

Here's the definition of ComplexSinusoid, with print statements to display intermediate results.

Ввод [4]:
```
from thinkdsp import Sinusoid

class ComplexSinusoid(Sinusoid):
    """Represents a complex exponential signal."""

    def evaluate(self, ts):
        """Evaluates the signal at the given times.

        ts: float array of times

        returns: float wave array
        """
        print(ts)
        phases = PI2 * self.freq * ts + self.offset
        print(phases)
        ys = self.amp * np.exp(1j * phases)
        return ys
```

Here's an example:

Ввод [5]:
```
signal = ComplexSinusoid(freq=1, amp=0.6, offset=1)
wave = signal.make_wave(duration=1, framerate=4)
print(wave.ys)
```

```
[0.   0.25 0.5  0.75]
[1.    2.571 4.142 5.712]
[ 0.324+0.505j -0.505+0.324j -0.324-0.505j  0.505-0.324j]
```

The simplest way to synthesize a mixture of signals is to evaluate the signals and add them up.

Ввод [6]:
```
from thinkdsp import SumSignal

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

Here's an example that's a mixture of 4 components.

Рис. 1: Запуск примеров

2 Упражнение №2

Во втором пункте седьмой лабораторной работы было продемонстрировано использование ДПФ (дискретное преобразование Фурье) и обратное ДПФ в виде произведения матриц. Такие операции занимают время $N2$, где `N` - длина массива, что достаточно быстро для большинства применений, но есть более быстрый алгоритм. Быстрое Преобразование Фурье (БПФ) или `FFT`, занимающий $Nlog(N)$. Ключевая вещь в БПФ это лемма Даниелсона-Ланцоша, 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-Ланцоша.

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.

Начнем с реального сигнала и вычислим его БПФ.

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

```

Листинг 1: Получение БПФ

Получившиеся значения: `[ 0.2+0.j -1.2-0.2j 0.2+0.j -1.2+0.2j]`

Реализуем функцию `dft` для вычисления матрицы синтеза и протестируем ее.

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

```

Листинг 2: Функция dft

```
hs2 = dft(ys)
np.sum(np.abs(hs - hs2))
```

Листинг 3: Использование dft

Получившиеся значение 5.864775846765962e-16.

Реализуем функцию `fft_norec`, которая разобьет входной массив и использует `np.fft.fft` для вычислеия БПФ.

```
def fft_norec(ys):
    N = len(ys)
    He = np.fft.fft(ys[::2])
    Ho = np.fft.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: Функция fft_norec

```
hs3 = fft_norec(ys)
np.sum(np.abs(hs - hs3))
```

Листинг 5: Использование fft_norec

Получившиеся значение 0.0.

Реализуем функцию `fft`. Она похожа на предыдущую, но в ней `np.fft.fft` заменено на рекурсию.

```
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 6: Функция fft

```
hs4 = fft(ys)
np.sum(np.abs(hs - hs4))
```

Листинг 7: Использование fft

Получившиеся значение 1.6653345369377348e-16.

В результате можно сказать, что полученная нами реализация БПФ занимает время, пропорциональное $Nlog(N)$ при создании и копировании массивов, а также занимает аналогичное количество места.

3 Выводы

В результате выполнения данной лабораторной работы были изучены понятия ДПФ, БПФ. Была создана функция для вычисления БПФ, которая работает за $Nlog(N)$ при создании и копировании массивов. Кроме того мы изучили примеры из файла `chap07.ipynb`, запустив все блоки кода и прочитав всю информацию.