Санкт-Петербургский государственный политехнический
университет Петра Великого

**Высшая школа интеллектуальных систем и
суперкомпьютерных технологий**

Отчёт по лабораторной работе №7
Дисциплина: Телекоммуникационные технологии
Тема: Дискретное преобразование Фурье

Выполнил студент гр. 3530901/80201 В.А. Пучкина

Преподаватель: Н.В. Богач

Санкт-Петербург
2021

Содержание

Список иллюстраций

Листинги

1 Упражнение 7.1

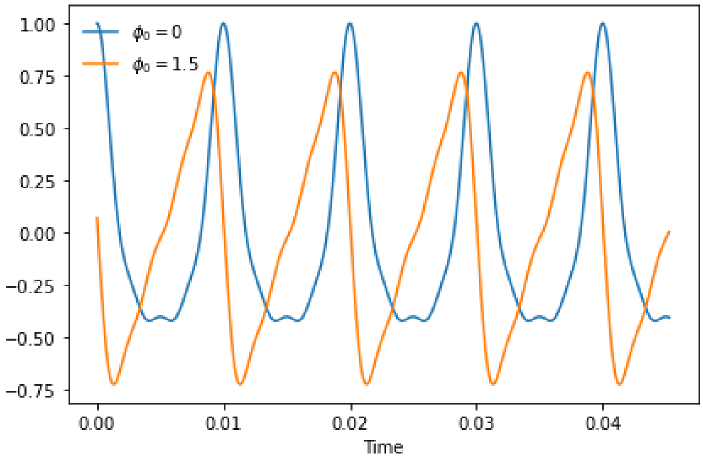
В этом упражнении необходимо изучить примеры в файле `chap07.ipynb`.

```
wave = Wave(ys.real, framerate)
wave.apodize()
wave.make_audio()
```

To see the effect of a complex amplitude, we can rotate the amplitudes by 1.5 radian:

```
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'$\phi_0 = 1.5$')
decorate(xlabel='Time')
```



Rotating all components by the same phase offset changes the shape of the waveform because the components have different periods, so the same offset has a different effect on each component.

Рис. 1: Изучение примеров.

Все примеры были запущены и изучены.

2 Упражнение 7.2

В данном упражнении необходимо реализовать быстрое преобразование Фурье (БПФ). Этот алгоритм позволяет ускорить преобразование Фурье с N^2 до $NlogN$. Алгоритм выглядит следующим образом:

1. Дан массив сигнала y. Разделим его на чётные элементы e и нечётные элементы o.
2. Вычислим ДПФ e и o, делая рекурсивные вызовы.
3. Вычислим ДПФ(y) для каждого значения n, используя лемму Дэниелсона-Ланцоша.

В простейшем случае эту рекурсию надо продолжать, пока длина y не дойдёт до 1. Тогда ДПФ(y) = y. А если длина y достаточно мала, можно вычислить его ДПФ перемножением матриц, используя заранее вычисленные матрицы.

Итак, сначала нам потребуется определить функции `synthesis_matrix` и `dft`, которые были использованы в примерах из § 1.

```
PI2 = numpy.pi * 2

def synthesis_matrix(N):
    ts = numpy.arange(N) / N
    freqs = numpy.arange(N)
    args = numpy.outer(ts, freqs)
    M = numpy.exp(1j * PI2 * args)
    return M
```

Листинг 1: Функция `synthesis_matrix`.

```
def dft(ys):
    N = len(ys)
    M = synthesis_matrix(N)
    amps = M.conj().transpose().dot(ys)
    return amps
```

Листинг 2: Функция `dft`.

Теперь возьмём небольшой сигнал и вычислим его ДПФ с помощью функции `numpy.fft.fft`.

```
ys = [-0.2, 0.6, 0., -0.5]
hs = numpy.fft.fft(ys)
print(hs)
```

Листинг 3: Вычисление ДПФ с помощью `numpy.fft.fft`.

```
[-0.1+0.j  -0.2-1.1j -0.3+0.j  -0.2+1.1j]
```

Листинг 4: Результаты для `numpy.fft.fft`.

А теперь сравним эти результаты с результатами, которые получатся при использовании функции `dft` (Листинг.2).

```
ys = [-0.2, 0.6, 0., -0.5]
hs = numpy.fft.fft(ys)
print(hs)
```

Листинг 5: сравнение результатов функций `dft` и `numpy.fft.fft`.

```
[-0.1+0.00000000e+00j -0.2-1.10000000e+00j -0.3+1.10218212e-16j -0.2+1.10000000e
   +00j]

6.230083922242757e-16
```

Листинг 6: Результаты сравнения.

Как видно из результатов сравнения, различия минимальны. А потому мы можем использовать `dft` в качестве «основы» для реализации БПФ.

Теперь приступим к реализации алгоритма. Начнём с функции, разбивающей входной массив на чётные и нечётные элементы и использующей функцию `numpy.fft.fft` для вычисления ДПФ полученных половин вместо использования рекурсивного вызова.

```
def fft_norec(ys):
    N = len(ys)
    He = numpy.fft.fft(ys[::2])
    Ho = numpy.fft.fft(ys[1::2])

    ns = numpy.arange(N)
    W = numpy.exp(-1j * PI2 * ns / N)

    return numpy.tile(He, 2) + W * numpy.tile(Ho, 2)
```

Листинг 7: Функция `fft_norec`.

Вычислим БПФ с помощью этой функции и сравним полученный результат с другой реализацией.

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

Листинг 8: Сравнение результатов функциий `fft_norec` и `numpy.fft.fft`.

```
[-0.1+0.0000000e+00j -0.2-1.1000000e+00j -0.3-1.2246468e-17j -0.2+1.1000000e+00j]

2.620466485321338e-16
```

Листинг 9: Результаты сравнения.

Как мы видим, разница всё ещё мала, что означает, что наша функция работает корректно.

Теперь заменим нерекурсивные `numpy.fft.fft` на рекурсивные вызовы и добавим базовый случай.

```
def fft(ys):
    N = len(ys)
    if N == 1:
        return ys

    He = fft(ys[::2])
    Ho = fft(ys[1::2])

    ns = numpy.arange(N)
    W = numpy.exp(-1j * PI2 * ns / N)
    return numpy.tile(He, 2) + W * numpy.tile(Ho, 2)
```

Листинг 10: Функция `fft` с рекурсивными вызовами.

Сравним результаты функции `fft` с результатами другой реализации.

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

Листинг 11: Сравнение результатов функций `fft` и `numpy.fft.fft`.

```
[-0.1+0.0000000e+00j -0.2-1.1000000e+00j -0.3-1.2246468e-17j -0.2+1.1000000e+00j]

4.0082452661027834e-16
```

Листинг 12: Результаты сравнения.

Как мы видим, полученная разница минимальна, а потому можно сказать, что написанная нами функция `fft` работает корректно. Более того, при такой реализации время выполнения составляет $NlogN$ (вместо N^2). Однако и требуемое пространство пропорционально $NlogN$. Кроме того, при такой реализации приходится тратить время на создание и копирование массивов.

В ходе выполнения данного упражнения был реализован алгоритм БПФ, время выполнения которого составляет $NlogN$ (вместо N^2). Функция была протестирована сравнением её результатов с результатами `numpy.fft.fft`. Разница мала, а потому был сделан вывод, что составленная функция работает корректно. Также были разобраны достоинства (быстрое выполнение) и недостатки (требуется пространство и время для работы с массивами) такой реализации.

3 Выводы

В ходе выполнения данной лабораторной работы были изучены дискретное преобразование Фурье (ДПФ) и быстрое преобразование Фурье (БПФ). Кроме того, была написана функция, реализующая алгоритм БПФ. Она была протестирована, и был сделан вывод, что она работает корректно. При выборе используемой реализации преобразования Фурье стоит обращать внимание на длину массива сигнала. Если она мала, то будет достаточно использовать ДПФ (работает за N^2), если же длина большая, то стоит обратить внимание на БПФ (работает за $NlogN$), чтобы уменьшить время вычисления. Однако при использовании конкретно этой реализации из § 2 важно помнить, что её требуемое пространство также пропорционально $NlogN$.